Department of Computer Science
Series of Publications A
Report A-2020-2

# Edge-Facilitated Mobile Computing and Communication

Pengyuan Zhou

*Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki, in Porthania P673, on the* $28^{th}$ *of May, 2020 at 12 o'clock noon.*

University of Helsinki
Finland

**Supervisor**
Jussi Kangasharju, University of Helsinki, Finland

**Pre-examiners**
George Polyzos, Athens University of Economics and Business, Greece
Ioannis Psaras, University College London, United Kingdom

**Opponent**
Tarik Taleb, Aalto University, Finland

**Custos**
Jussi Kangasharju, University of Helsinki, Finland

**Contact information**

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911


ISSN 1238-8645
ISBN 978-951-51-6151-2 (paperback)
ISBN 978-951-51-6152-9 (PDF)
Helsinki 2020
Unigrafia

**Edge-Facilitated Mobile Computing and Communication**

Pengyuan Zhou

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
pengyuan.zhou@helsinki.fi
http://zpymyyn.github.io

PhD Thesis, Series of Publications A, Report A-2020-2
Helsinki, May 2020, 137 pages
ISSN 1238-8645
ISBN 978-951-51-6151-2 (paperback)
ISBN 978-951-51-6152-9 (PDF)

**Abstract**

The proliferation of IoT devices and rapidly developing wireless techniques boost the data volume and service demand at the edge of the Internet. Meanwhile, increased requirement for low latency feedback has become a must for most popular mobile applications, e.g., Augmented Reality (AR), Virtual Reality (VR) and Connected Vehicles. To address these challenges, edge computing has emerged as an extensional solution for cloud computing.

This thesis studies edge computing-facilitated mobile computing and communication systems. We first propose solutions to improve edge resource utilization regarding general edge systems. We present a mechanism to cluster user requests based on similarity for better Content Delivery Network (CDN) performance. This mechanism works directly on current CDN architecture and can be deployed incrementally. Then we extend the mechanism by adding cache resource grouping algorithm, so that the system directs similar requests to same servers and group those servers which receive similar requests. This iterative mechanism optimizes the edge utilization by concentrating the resource on similar requests to achieve higher cache hit ratio and computation efficiency.

Thereafter, we present solutions for mobile edge systems specifically for three most promising use cases, i.e., Connected Vehicles, Mobile AR (MAR) and Smart city (traffic control). We explore the potential of edge computing

in connected vehicular AR applications with real data sets. We design a lightweight edge system and data flow fit for general connected vehicular AR applications and implement a prototype. With an indoor test and real data set analysis, we find out that our system can improve the performance of vehicular AR applications with reasonable cost. To optimize the system, we formulate the problem of edge server allocation and task scheduling as a mutant multiprocessor scheduling problem and develop a two-stage edge-cloud decentralized algorithm as well as a centralized algorithm to schedule the offloading tasks on the fly. We conduct a raw road test and an extensive evaluation based on the road test results and large data sets from real world. The results show that our system improve at least twice the application performance comparing with cloud solutions.

For MAR, we consider to offload tasks to multiple edge servers via multiple paths simultaneously to further improve the MAR performance. We develop a fast scheduling algorithm to split the workloads among the available edge servers and show promising results with real implementations. At last, we explore the potential of combining edge computing and machine learning techniques to realize intelligent traffic control by letting edge servers co-located with traffic lights learn the waiting traffic and adapt the light periods with reinforcement learning.

**Computing Reviews (2012) Categories and Subject Descriptors:**

Computer systems organization → Distributed architectures
Networks → Network services
Networks → Network performance evaluation

**General Terms:**
Design, Implementation, System, Performance, Experimentation

**Additional Key Words and Phrases:**
Edge computing, Mobile computing, Connected vehicles, Augmented reality, Task scheduling, Offloading

# Acknowledgements

Educated in Asian culture, competing with millions of students (which is not an overstatement), I always aimed at high goals and big questions. I still held my mindset during the first two years of my PhD, until the style and passion for scientific research of my supervisor, Professor Jussi Kangasharju, influenced my perspecitve. I still remember the day vividly, we were sitting and chatting in a couch and I asked, "what topic would you do in the next 5 years or near future", and Jussi said, "not sure about the exact topics, but gotta be something interesting". That answer got me immediately. It was a genuine, simple, straightforward motivation for research, which I almost forgot at that moment due to focusing all on competition. And that chat, along with Jussi's passionate, totally change my view of research and shapes the angle of how I pursue research thenceforce. Therefore, I would like to express my sincere gratitude to Jussi. The help he gave me is more than any concrete instructions or guides, which he also provided lots, but the perspective of what to do as a researcher. And I truly believe that mindset also guides me in other aspects of my life.

Another game changer for my research is the visit to HKUST in 2018 for nearly 5 months. The host, Professor Pan Hui and the members of Symlab especially Tristan Braud, Wenxiao Zhang and Xiang Su motivate me to work much harder since then with their extreme hard working style. Also, their works on mobile and AR inspires me and gives me an opportunity to realise how much I am interested in those fields, and they are of my major research topics nowadays.

I also would like to thank all the members in CoNe lab, Aleksandr Zavodovski, Nitinder Mohan, Ossi Karkulahti, Otto Waltari, Suzan Bayhan, Walter Wong, all of whom have created such a friendly working environment. I will always remember the numerous brainstorms during the weekly group meetings. I also owe my gratitude to Pirjo Moen, who has been providing support since day one of my PhD program. Her advices and opinions on study and life are really helpful especially for foreign students like me.

Last but not least, I would like thank all the seniors who have supported my research and study, especially Professor Xiaoming Fu from University of Göttingen and Professor Otto Wittner from NTNU. Professor Xiaoming is the PI of the biggest project I have been involved by far. He is really nice and has given me lots of great advices and helps. Professor Otto was the host of my research visit to Trendheim, Norway in 2017. He and his colleagues in UNINETT helped me in a fundamental way of coding and working styles. I am very lucky to be involved and funded by the EU FP7 Marie-Curie-Actions project, CleanSky, for the first three years of my doctorate program. I also want to thank the Academy of Finland projects, WMD and AIDA, which supported the rest of the program.

I would like to express my deepest gratitude to my parents, who have always been supporting me in every way they can. I have never stopped learning from them, even now I know I still have a long way to go to become a good person like them. And also my love, Ada, who has changed my attitude towards life and work from "well, okay" to "I will make it". I can not imagine doing all these without her and she is the most important person who helped me recognise what I really want to do, not only now but also the future.

I can not mention everyone here, but thank you all who have been there for me.

Helsinki, May 2020
Pengyuan Zhou

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The rapid development of networking technologies pushes forward the proliferation of mobile systems, e.g., smart home, smart city and automotive vehicle etc., or the Internet of Things (IoT) in more generalized terms. On top of the modern IoT systems, countless applications are bursting into the market to provide services in various fields. The numerous applications generate large amounts of data from end users everyday, resulting in the heavy tail of data flow in the Internet drifting from the central cloud towards the edge of the network.

The tremendous amount of data generated by many mobile applications requires real-time feedback. For instance, Augmented Reality (AR), Virtual Reality (VR) and vehicular communication applications require user experienced latency to be lower than 100 milliseconds in general and lower than 20 milliseconds in some scenarios [1]. The majority of academic and industrial research institutes has spent a huge amount of effort on improving the transmission speed in core network and processing efficiency in cloud data centers. Some proposals bring out satiable results especially in wired networking transmissions.

The big problem remaining is the last mile transmission latency in wireless communication systems. The last hop from a user to the first node in the ISP network has recently been identified as the major contribution to the overall wireless transmission delay [2]. Current 4G networks meet the demand of traditional applications like web browser, video streaming and video call etc. However, when the latency requirement becomes under 100 ms or even 20 ms, alternative solutions need to be explored.

One straightforward solution, Edge Computing, is to deploy low cost servers adjacent to the end users on the edge of the network. While deploying edge servers in a wired networking environment seems simple, e.g. the edge cache servers that are widely deployed in most CDNs nowadays,

it becomes much more problematic when we want to further explore the potential of edge computing. For instance, how to improve the utilization efficiency of edge servers from a general point of view regardless of wired or wireless networking is complicated. Unlike central data centers in which servers are deployed and connected closely, edge servers are deployed in a distributed manner. As such, to improve the utilization efficiency one needs to consider not only the edge servers but also their interactions with end users and the remote cloud data centers. Moreover, the distributed deployment brings up fundamental problems such as where to deploy edge servers in wireless environment and how to devise feasible systems to incorporate edge servers into the surrounding environment to optimize the system performance. In other words, while cloud data center is more an indoor problem focusing on designing a huge warehouse accommodating tens of thousands of servers, edge servers are envisioned to be deployed in outdoor environments for many use cases and thus need to address the concern of interaction and cooperation with circumambient objects that may affect the system. Now we briefly describe the conceptual flow of this thesis and outline detailed research questions in the following section.

At the beginning of the thesis, we solve the efficient utilization of edge resources by addressing two requirements it should meet with. First, it should cache as many popular contents as possible based on a historical request track. Second, which is harder to meet, is to redirect the same requests to same cache servers for a higher cache hit ratio, which reduces uplink transmission and user experience latency potentially. While a number of caching policies exist to address the first requirement, solutions to meet the second are yet to be explored. One of our approaches is to cluster user requests based on similarity. Another approach is to group the edge servers according to their received requests for better grouped resource utilization. These solutions look at edge computing from a general perspective by focusing on improving system performance regardless of specific scenarios and system features. While it does provide some beneficial results, we dig deeper into practical problems to address the details such as deployment and system design for different kinds of use cases.

To deploy the edge system in the existing environments such as urban area or buildings, the design needs to consider a number of influential factors from different points of view. For instance, one of the most well-known standards of edge deployment in urban areas is Mobile Edge Computing (MEC) developed by the European Telecommunications Standards Institute. The basic idea of the standard is to deploy edge servers at the base stations. It has advantages of reusing existing facilities including

the cellular tower, power support and core network connection. However, it also faces challenges such as the last mile latency as mentioned before. Moreover, the challenges that edge systems face vary in different use cases. Therefore, each edge system, besides the basic standard technologies, requires dedicated design to integrate itself into the surrounding environment and fulfill specified user requirements. This is a major difference between the design of an edge system and a central data center, since the latter cares more about the inside of the center instead of the surroundings. The middle part and latter part of the thesis focuses on these problems in different use cases.

## 1.1 Research Questions

This thesis focuses on how to use edge computing to facilitate mobile computing and communication systems and specifically targets the following research questions:

1. How do we improve the utilization efficiency of edge resources in CDN and general edge cloud system (Chapter 3 and Chapter 4)?

2. Can edge computing facilitate vehicular AR applications (Chapter 5)?

3. How do we design the edge system, the function flow and deploy the edge servers for vehicular AR applications (Chapter 6)?

4. Can multi-path and multi-server offloading improve the performance of MAR applications? And how (Chapter 7)?

5. How do we combine machine learning and edge computing to get better city traffic control (Chapter 8)?

In this thesis, we present several works to address the described questions with contributions outlined in the following section.

## 1.2 Thesis Contributions

The first contribution (Chapter 3) proposes to profile and group users according to their interest profiles. We consider edge caching as an example and show the potential benefits of directing users from the same group to the same caches through evaluation. We investigate a range of workloads and parameters and the same conclusions apply. Our results highlight the

importance of grouping users and demonstrate the potential benefits of this approach.

The second contribution (Chapter 4) develops a solution to predict and store data in edge resource caches for upcoming computations based on existing edge and fog computing models. Our solution is based on grouping caches according to the workloads they serve. We further develop methods for populating the caches and ensuring the coherence of the cached data.

The third contribution (Chapter 5) is a novel Vehicle-to-Edge (ARVE) system for AR applications in vehicular networks. With computational units co-located with the base stations and aggregation points, the system embeds computation at the edge of the network allowing to reduce the overall latency compared to vehicle-to-cloud and significantly trim the complexity of vehicle-to-vehicle communication. We also propose an envisioned use case, connected ARHUD (Augmented Reality Heads-Up-Display) and a preliminary simulation of edge server deployment.

The fourth contribution (Chapter 6), EARVE, is a detailed systems design following the use case proposed in Chapter 3. Based on Linux Foundation project EdgeXFoundry, we present a layered edge system and deploy an edge server prototype. We test the connected ARHUD with indoor experiment setup to prove the low latency of an edge system in real-time emergency applications. We also investigate the scalability of EARVE using real traffic data from London and show its benefit in realistic scenarios for different traffic densities.

The fifth contribution (Chapter 7) provides an extension to current mobile edge offloading models using multiple paths specified for MAR applications. We present a model for multi-server device-to-device, edge and cloud offloading. We also propose a new task allocation algorithm exploiting this model for MAR offloading. With the prototype and tests, we show that multipath offloading with our allocation algorithm outperforms other multipath offloading without dedicated allocation algorithm or single server offloading.

The sixth contribution (Chapter 8) looks into the potential of combining the technology of edge computing and machine learning, i.e. the so-called Edge AI. It proposes ERL, a solution based on edge computing nodes to collect traffic data and alleviates congestion by providing optimized traffic light control in real time. Edge servers run fast reinforcement learning algorithms to tune the metrics of the traffic signal control algorithm ran in each intersection. ERL operates within the coverage area of the edge server, and uses aggregated data from neighboring edge servers to provide city-scale congestion control.

## 1.3 Thesis Organization

The thesis first introduces an overview of edge computing development, followed by six research chapters addressing the challenges and a conclusion in Chapter 9.

In Chapter 3 and Chapter 4, we focus on research question 1. The works look at improving edge resource utilization from different angles, i.e., redirecting user requests to the same servers for higher cache hit ratio and group edge resource based on received requests for higher grouped resource utilization efficiency, respectively.

Chapter 5 explores the possibilities of edge computing in real-time connected vehicle applications (research question 2). Specifically, it proposes a new type of application based on real-world technique development, i.e., Connected Vehicular Vision using ARHUD. With the proposed system design and primary evaluation, it shows the potential of edge computing in vehicular AR systems. Following the futuristic use case proposed in the previous chapter, Chapter 6 proposes the detailed system design, edge server deployment solution, real world prototype, road tests and extensive simulation results (research question 3). The chapter looks through the detailed problems from different aspects of edge-facilitated connected vehicle system and propose solutions on different levels, e.g., system functionality design, data flow, infrastructure deployment and offloading etc.

The last two research chapters, Chapter 7 and Chapter 8, consider problems from two very different angles. As multipath TCP increases in popularity, offloading traffic via multipath seems to hold a future for better networking performance. Chapter 7 steps forward along this trend and examines the MAR performance when offloading to multiple servers via multiple paths. Chapter 8 focuses on the intelligent traffic system and proposes an edge AI solution to improve traffic light control with reinforcement learning.

Each research chapter is self-contained and introduces an overview of related works with a looking at its future directions. Finally, Chapter 9 concludes the thesis with summarized contributions.

## 1.4 Publications

The works in Chapters 3, 4, 5, 6, 7 and 8 have been published in the following publications, respectively:

**Chapter 3.** P. Zhou and J. Kangasharju. Profiling and Grouping Users to Edge Resources According to User Interest Similarity. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking (CAN '16)*, Irvine, CA, USA, December 12, 2016.

**Contribution:** The publication was led by the author who proposed the methodology and designed the solution algorithm. The author also evaluated the proposal with simulation. Prof. Jussi Kangasharju contributed to the planning and discussion of idea formulation and evaluation setup. The author and Prof. Jussi Kangasharju were involved in the writing process of the publication.

**Chapter 4.** N. Mohan, P. Zhou, K. Govindaraj, and J. Kangasharju. Managing Data in Computational Edge Clouds. In *Proceedings of the Workshop on Mobile Edge Communications (MECOMM '17)*, Los Angeles, CA, USA, August 21, 2017.

**Contribution:** The publication was led by Nitinder Mohan who formulated the problem, designed the algorithm, did the evaluation and writing of the final publication. The author contributed to the discussion of problem formulation and algorithm design, implementation and simulation setup. Keerthana Govindaraj provided critical insights regarding related work and scope driven by her personal industrial experience and also contributed to improving the writing and paper structure. Prof. Jussi Kangasharju was involved in the planning, discussion and writing process of the article.

**Chapter 5.** P. Zhou, W. Zhang, T. Braud, P. Hui and J. Kangasharju. ARVE: Augmented Reality Applications in Vehicle to Edge Networks. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Mobile Edge Communications Networking (MECOMM '18)*, Budapest, Hungary, August 20, 2018.

**Contribution:** The publication was led by the author who proposed the use case, designed the fundamental system architecture and solution for deployment. The author also collected a real-world dataset for evaluation. Wenxiao Zhang was the major contributor of the evaluation. Dr. Tristan Braud outlined the challenges and was involved in the discussion of system design and writing process. Prof. Pan Hui was involved in the discussion of idea formulation. Prof. Jussi Kangasharju contributed to the discussion of idea formulation and the writing process of the article.

**Chapter 6.** P. Zhou, W. Zhang, T. Braud, P. Hui and J. Kangasharju. Enhanced Augmented Reality Applications in Vehicle to Edge Networks. In *Proceedings of the 22nd Conference on Innovation in Clouds, Internet and Networks (ICIN 2019)*, Paris, France, 19-21 Feb. 2019.

**Contribution:** The publication was an extension of a previous publication, led by the author who designed the detailed system functionalities and solution for deployment. The author also collected the real-world dataset and implemented part of the system prototype. Wenxiao Zhang was the major contributor of the implementation and evaluation setup. Dr. Tristan Braud presented the motivation and was involved in the discussion of functionality design and writing process. Prof. Pan Hui was involved in the idea discussion and formulation. Prof. Jussi Kangasharju contributed to the overall paper direction, paper structure refinement and writing improvement.

**Chapter 7.** T. Braud, P. Zhou, J. Kangasharju and P. Hui. Multipath Computation Offloading for Mobile Augmented Reality: Latency, Scheduling and Optimizations. *In Proceedings of the IEEE International Conference on Pervasive Computing and Communications (Percom 2020).*

**Contribution:** The publication was led by Dr. Tristan Braud who formulated the problem, designed the algorithm, did the simulation and writing of final publication. The author was involved in the discussion of algorithm design, especially considering the practical factors. The author also outlined the system data flow, built the prototype selecting a proper usecase and evaluated the proposal with the implementation. Prof. Jussi Kangasharju was involved in the discussion of the article. Prof. Pan Hui was involved in the discussion and refinement of the paper structure.

**Chapter 8.** P. Zhou, T. Braud, A. Alhilal, P. Hui and J. Kangasharju. ERL: Edge Based Reinforcement Learning for Optimized Urban Traffic Light Control. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Kyoto Japan, March 2019.

**Contribution:** The publication was led by the author who proposed the use case, designed the fundamental system architecture and the tiered algorithm. The author also collected the real-world dataset and was the major contributor of the evaluation. Dr. Tristan Braud outlined the challenges and was involved in the discussion of system design and writing process. Ahamed Alhilal made important contributions to the evaluation setup. Prof. Pan Hui was involved in the discussion of idea formulation. Prof. Jussi Kangasharju contributed to the discussion of idea formulation, evaluation setup and the writing process of the article.

# Chapter 2

# Overview of Edge Computing

In this chapter we give an overview of edge computing. First we discuss the relation between edge computing and cloud computing and recap why we need it. Then we select several major advantages to show the benefit we get from the edge solutions. Thereafter, some of the most attractive use cases are presented to show the future potential of edge computing in various fields. At last, we outline the current development with the most advanced projects and conclude this chapter.

## 2.1 Cloud and Edge

Cloud computing has dominated the market over the past decade. It provides on-demand computing and storage resources available to the end users without requiring direct active management as shown in Figure 2.1. Usually cloud computing decomposes the huge calculation programs into countless small programs and then processes and analyzes these small programs through a system composed of numerous servers to get the results and return them to the user [3]. Users pay the service fee on-demand and do not need to maintain the required resources such as hardware, platform and software. For a decade, it has attracted so many users and demands that most data processing happens in the cloud nowadays. The development of networking techniques (4G and SDN etc.) and data processing techniques (machine learning etc.) makes the cloud even more attractive than ever. Meanwhile, the decreasing price of hardware also contributes to the continuous growth of the cloud market.

However, the increasing wireless transmission speed and the processing power in end user devices promote the volume of floating data on the edge of the Internet, hence adding burden to the network between end users

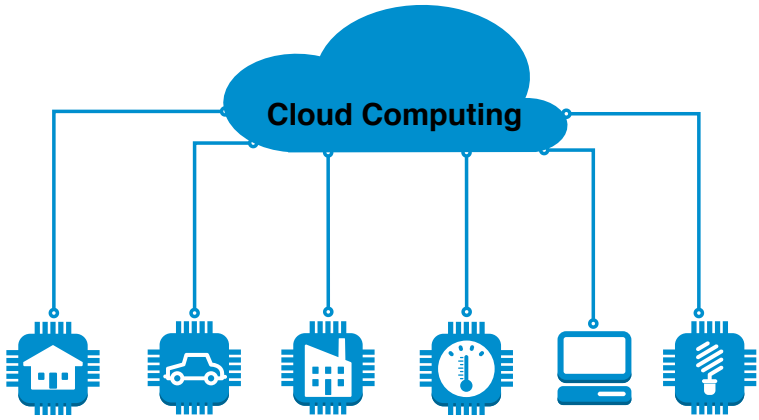



Figure 2.1: Architecture of cloud computing.

and the cloud. For instance, 5G, which has started to be deployed in some countries, targets at 1 ms latency [4]. Although most likely not every use case can achieve that goal in the end, there is still a good chance 5G will provide comparable transmission speed with cable network, e.g. 10 ms latency for the last mile transmission. What would happen then? One possible result, among several others, is that the last mile transmission becomes so fast that it becomes more reasonable to let the local servers do the computation to avoid the upload latency, which becomes cumbersome. We will answer this question in more details in the next section. Besides, mobile applications that are *computational intensive* and *latency sensitive* are getting more and more popular nowadays. For example, VR and AR both demand complex computation, generate large amounts of data and are very sensitive to latency [5]. As such, most solutions nowadays choose to process the data locally instead of uploading to the cloud to avoid the high latency and big bandwidth requirement.

Since Amazon first promoted its "Elastic Compute Cloud" in 2006, a new computing paradigm, edge computing, arises as an extensional solution for the cloud which places substantial computing and storage resources at the edge of the Internet as shown in Figure 2.2. This concept traces back to 1999 when an MIT team founded Akamai and introduced CDNs to accelerate web performance by deploying edge servers close to the end users to prefetch and cache web content [6, 7]. Being in close proximity to the end users, edge computing generalizes the concept of CDN by adding complicated processing power to the storage capacity and has the potential to improve the response time, transmission bandwidth cost, as well as data safety and user privacy [8]. Although the distributed deployment

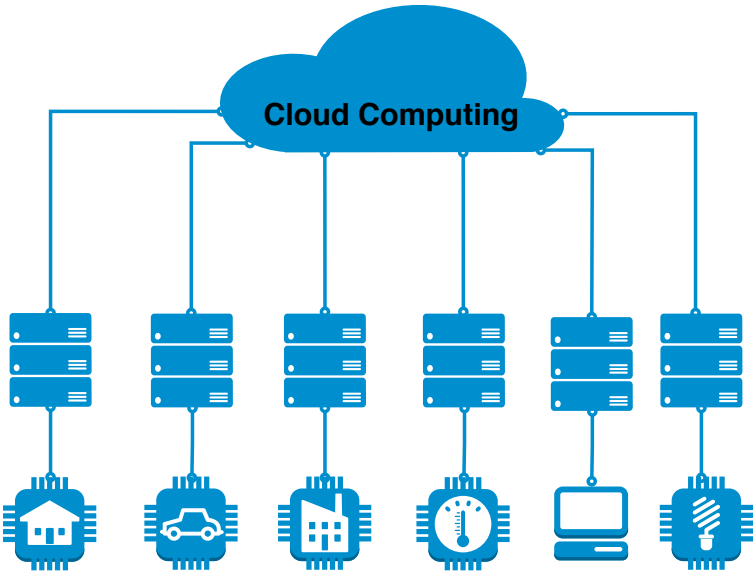



Figure 2.2: Architecture of cloud-edge computing.

seems to require more manual support, edge computing can reduce the cost compared to cloud computing. A cloud data center consisting of tens of thousands of servers composes a highly complicated inner network. To perform well, the data center needs to maintain a large number of facilitating servers for the function of gateway, caching, queuing, monitoring and self-maintenance etc. Edge computing, on the other hand, serves a smaller scale of services and users, thus requiring a much smaller number of servers without the demand for lots of facilitating servers. In the next section, we will elaborate on more motivations of edge computing with some major benefits.

## 2.2 Motivations of Edge Computing

Edge computing complements cloud computing by providing quite a few advantages. In this section, we outline the three most attractive benefits to elaborate the capability of edge computing, i.e., short *latency*, low *traffic volume*, and less *bottleneck*.

**Latency:** The most straightforward advantage is the improvement of latency, thanks to the proximity of edge servers to the end users. Most edge solutions currently propose to deploy the servers at one or two hops away from the end user, being controllers in smart homes (e.g. Google as-

sistant), servers co-located with base stations, or road-side units for smart city etc. Besides the short distance transmission, the developing wireless techniques also help edge computing to hold a better future. The most relevant techniques include 5G, Wi-Fi 6 and Dedicated Short Range Communications (DSRC), each of which facilitates edge computing in a representative communication environment, i.e., cellular network, Wi-Fi and Machine-to-Machine (M2M) [9, 10]. Since proposed in 2008 by NASA and Machine-to-Machine Intelligence (M2Mi) Corp, 5G has attracted lots of attention and now has started to be deployed by lots of countries all over the world. The extremely low latency of last mile transmission guaranteed by 5G will enable numerous futuristic applications that cannot function well today, e.g., automotive and AR. Wi-Fi 6, i.e. IEEE 802.11 ax, promoted by Wi-Fi Alliance, is a high efficiency wireless communication protocol. IEEE 802.11ax is designed to operate between 1 and 6 GHz and aims at providing 4 times the throughput of IEEE 802.11ac at the user layer with only 37% higher nominal data rates at the PHY layer [11]. DSRC, on the other hand, is a set of protocols and standards designed specifically for connected vehicular technology. Its feature of short-range fast transmission has attracted lots of academic and industrial attention to help realize high-speed wireless communication between vehicles and road-side infrastructures. These wireless techniques alongside others help edge computing achieve extremely low latency in different networking environments.

**Traffic:** As edge servers provide considerable processing and storage capacity, only the data that requires further processing, backup or fetching needs to be uploaded to or downloaded from the cloud, resulting in lower volumes of network traffic. This advantage has been proved by the major video streaming providers, e.g. Open Connect used by Netflix, Google Cloud CDN used by Youtube and CloudFront used by Amazon Prime Video. These solutions localize substantial amounts of traffic by deploying edge cache servers close to the end users all over the world. Besides content delivery, this advantage also benefits computation offloading especially facing the IoT era, in which tremendous IoT sensors and devices are connected to the Internet and transfer explosively increasing data for processing. Currently, most IoT data is sent to remote cloud servers and cause expensive transmission cost. Edge computing mitigates the cost by offloading significant amounts of data to local servers.

**Bottleneck:** By de-emphasizing the role of the cloud, edge computing eases the network bandwidth demand and end user's reliance on the cloud thus alleviating the bottleneck and single point of failure.

## 2.3 Potential Use cases

Edge computing provides the greatest help for latency sensitive and computational intensive applications, by offloading data processing to a local server to avoid the latency to the remote cloud. Among numerous applications, we select four as the representations to elaborate the potentiality of edge computing, i.e. *connected vehicles*, *mobile AR/VR*, *smart city* and *smart factory*.

**Connected Vehicles:** Being a vehicle equipped with a self-driving system or an advanced driver-assistance system (ADAS), the fundamental requirement is the ability to be connected with other vehicles and road-side units (RSUs). Academic researchers and industrial companies have spent lots of effort to develop the communication protocol stack specifically for connected vehicles, i.e., DSRC. Together with 5G, vehicles are envisioned in the near future to communicate with each other and RSUs with extremely low latency and thus can exchange information in real time. Based on the connectivity, future autonomous and advanced vehicles demand valuable information from their own sensors, nearby vehicles and RSUs including traffic lights, street cameras and embedded sensors in the devices held by the pedestrians and bicyclists. The various and complicated data floating between the devices demands dedicated pre-processing before directly sending to the vehicles. And this is where edge computing comes into the play.

Equipped with different kinds of sensors and wireless communication systems, an autonomous vehicle intelligently detects and collects surrounding information in real time. A vehicle as such generates 4TB of raw data every hour according to a test by Intel, which is too large a volume of data to be all transferred to the cloud for processing [12]. On the other hand, co-located with RSUs, edge servers gather information and events from the neighbourhood and broadcast only the important ones to nearby vehicles. Furthermore, nearby devices can offload comprehensive tasks such as object detection and face recognition to edge servers for faster data processing instead of sending to the cloud. Compared to a cloud data center, edge computing provides faster data processing and information dissemination, which is crucial for driving safety.

**Mobile AR/VR:** Standalone mobile devices such as smartphones and AR/VR glasses provide poor user experience due to limited computation power and battery capacity nowadays. Both edge computing and cloud computing can provide offloading services. However, AR/VR applications generate a large amount of data and require real-time response, e.g., under 20 ms of end-to-end latency for a general mobile AR game to guarantee

authentic user experience or otherwise the virtual scene became unrealistic and even annoying. Besides facing larger transmission latency, cloud computing also needs to tackle the challenges of congested upload traffic and requirements for high bandwidth during peak hours. Take Pokémon Go Fest in Chicago, USA, 2017 as an instance [13]. It was a sold-out, day-long official event launched by the famous AR game, to let participants search for rare Pokémon. It drove huge crowds of 20,000 users to one park and strained the cellular network bandwidth soon after the event started. The connectivity became such a big problem that the game ended up unplayable for attendees. Edge computing can fix this issue by offloading the user requests to multiple local servers and thus reduces the upload traffic and eases the demand for bandwidth and processing capacity for core network and cloud, and improves the performance by providing low latency response and a user-friendly experience.

**Smart City:** Smart cities mainly include smart buildings, intelligent traffic control, and video surveillance. Edge computing can collect and analyse data in real time for better city operation. For example, sensors can be installed on streetlights to collect environmental data such as air quality and light intensity, for the sake of timely notification of air pollution and streetlight maintenance. Currently, most cameras on the roads do not have computation capacity, and thus have to transfer data to the cloud data center for image and video processing and data analysis. It works fine for long periods of traffic monitoring, but can not realize real-time feedback in milliseconds. An edge server with video campera and GPU is capable of processing the captured images for face recognition and object detection in real time for the use case such as intelligent traffic systems, e.g., identifying dangers and notifying nearby vehicles to avoid accidents, or completely real-time tracking of criminal vehicles.

**Smart Factory:** With the development of automation, factories are trying to further decrease the manpower cost by promoting smart factory to leverage the capabilities of connected sensors and devices for automated workflow tracking and adjustment. For instance, a smart factory can use intelligent edge gateways to collect local data and perform real-time filtering, or implement an industrial virtualized controller in the edge server to centralize the control of production line robotic arms. Since in these cases even a few milliseconds of lag can be a problem, it is worth trying to cut down the latency to the cloud. By deploying edge servers, functions that used to run on remote cloud servers can run physically closer to the end user, thereby cutting the distance to a server, which lowers the feedback latencies.

## 2.4 Development

Industry companies and academic researchers have had dramatically growing interest in edge computing in recent years. Efforts have been made in different dimensions to push forward the development of edge computing. We outline the state-of-the-art development as follows.

**International association:** European Telecommunications Standards Institute (ETSI) developed mobile edge computing (MEC), a network architecture concept that enables cloud computing capabilities and an IT service environment at the edge of the cellular network in 2014 [14]. The following year, the Open Edge Computing initiative was launched to shape the global ecosystem around edge computing. The members include Carnegie Mellon University (CMU), Intel, Microsoft, Nokia, NTT, T-Mobile, Vmware, Vodafone, Seagate and Crown Castle. In 2019, eighteen vendors and organizations have signed a cooperation agreement to form the Edge Computing Consortium Europe (ECCE). The consortium aims at creating a standard reference edge architecture and technology stack for industrial IoT domains. The major members include Huawei, Arm, Bombardier, B&R Automation, IBM, Intel, KUKA, Schneider Electric and Software AG etc.

**Commercial Services and Hardwares:** Several major cloud providers have launched edge computing projects to provide edge services, e.g., Microsoft's Azure IoT Edge, Amazon's IoT Greengrass, Google's Cloud IoT Edge, Cisco IOx application environment and IBM Edge Computing. Some giants also release hardware products specifically for edge computing purposes, e.g., Google Edge TPU, Dell EMC's Edge Gateways and HPE Edgeline series etc.

**Open Source Platform:** Open source platforms are developing well and have attracted researchers and companies to contribute. In early 2019, Linux Foundation has launched LF Edge, an umbrella organization aiming at establishing an interoperable open source framework for edge computing. The organization started with five projects and has grown to seven as for now, including Akraino Edge Stack, Baidu Baetyl, Edge Virtualization Engine, EdgeX Foundry, Fledge, Samsung Electronics Home Edge, and Open Glossary of Edge Computing. Other fast developing projects include Eclipse Kura, OpenStack StarlingX and Huawei KubeEdge etc.

## 2.5 Conclusion

This chapter has presented an overview of how edge computing has risen and evolved in the past years. Starting off as a successor of cloud comput-

ing, edge computing has received tremendous attentions from academia and industry and developed its own ecosystem now. Based on the fundamental advantage of proximity to users, edge computing has great potential in various fields with promising benefits. Among all the possibilities, four use cases hold the highest potentials, i.e. Connected Vehicles, Mobile AR/VR, Smart City and Smart Factory as described in Section 2.3. This thesis proposes relevant system designs and implementations for the former three use cases in Chapters 5, 6, 7 and 8.

# Chapter 3

# Request Clustering

## 3.1 Overview

Cloud computing provides a shared pool of resources for large-scale distributed applications. Recent trends such as fog computing and edge computing spread the workload of clouds closer towards the edge of the network and the users. Exploiting the edge resources efficiently requires managing the resources and directing user traffic to the correct edge servers. In this chapter we propose to profile and group users according to their interest profiles. We consider edge caching as an example and through our evaluation show the potential benefits of directing users from the same group to the same caches. We investigate a range of workloads and parameters and the same conclusions apply. Our results highlight the importance of grouping users and demonstrate the potential benefits of this approach.

## 3.2 Introduction

Cloud computing provides a shared pool of resources tackling large-scale distributed applications. With the introduction of SDN, NFV, CDNs, etc., controlling functions are becoming centralized while service and data are pushed towards the edge. These technologies allow for the provision of flexible and easily configurable control functions, and furthermore improve scalability and availability of data and services near the users. Recent approaches, such as fog computing [15–17] and edge computing [8], attempt to formalize the structure of how resources at the edge can be exploited for data and service provision.

A major concern is the efficiency of use of edge network resources. As argued in [18], clouds in cloud computing have different properties and

users and applications have different requirements. Hence, it may be hard to profile the cloud with a single explicit resource provisioning policy; besides, most related researches focus on top-down solutions or ignore features and requirements of end users. For instance, SDN and NFV provide high level network control functions mostly in the cloud. CDNs distribute content towards end user without analyzing the characteristics and profiles of users. To realize the full potential of edge network resources, we also need to profile user requests. The analysis and profiling of user request can help us free up network resources for a more targeted provisioning policy. Although users have heterogeneous and dynamic quality of service requirements, their interests are likely to be relative steady. Users normally take a long time to develop an interest which would last a long time in most cases; these interests change very slowly, at least relative to the frequency of incoming requests. Nowadays the popularity of recommender system may also accelerate the formulation of user interest and steady it. As studied in [19] and [20], recommender system can influence user preferences. According to the anchoring theory, user choice can be heavily influenced by the first piece of information offered [21]. Based on the above, we believe user interests can act as a means of request profiling.

In this chapter, we choose content delivery as an example of a cloud-backed service to illustrate the benefits of profiling users based on their interest preferences. According to the forecast of Cisco, IP video traffic will be 80 percent of all IP traffic by 2020 [22]. Also, the continued expansion of social networks brings lots of user-generated content. Content delivery is the main cloud-backed network services today and not only dedicated CDN providers like Akamai but also large companies like Google and Facebook are running CDNs.

CDNs are built around caches towards which user requests are directed. Research in the past has focused on caching hierarchies, cache replacement algorithms, and cache partitioning, but since these do not take any stand on how users are directed, they must act in a user-agnostic manner. Normally, users are directed to the closest cache in order to minimize network latency and traffic. However, users in the same area might have very different interest profiles, e.g., based on age, education, hobbies, and normal redirection mechanisms are unable to capture these. All content requested by the users competes for space in the cache. From an overall system efficiency point of view, this approach has two shortcomings:

1. It is unfair for non-mainstream users whose requests have smaller chances to be cached at the edge due to lower popularities. The requested content may be found in the parent cache with higher latency.

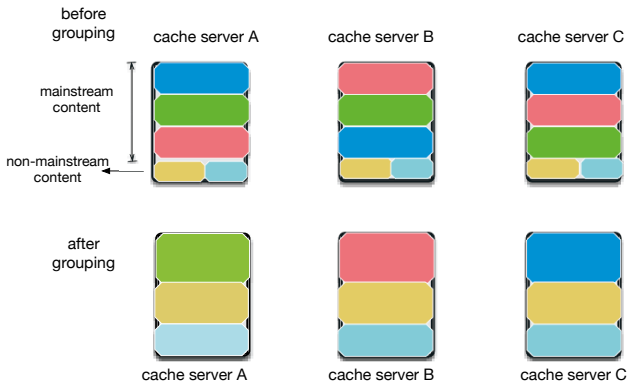



Figure 3.1: Difference of cache server with or without grouping users.

2. There can be a lot of duplicates of same contents in different cache servers. It results in waste of cache servers resource.

This motivates us to propose profiling requests and re-directing users based on interest similarity. This solution profiles traffic on a higher level of abstraction, comparing with profiling based on individual specific contents. Figure 3.1 shows an example where servers A, B, C and D cache lots of same contents since users in different areas share same interests. With current content popularity based cache strategy, the edge servers are more likely to cache mainstream contents. Since the cache size is limited, non-mainstream users have less chance to utilize the edge cache servers.

Considering the reasons above, we propose grouping users according to interest similarity to improve cache performance and efficiency. As shown in the lower part of Figure 3.1, if we can group users with similar interest and redirect their requests to same cache servers, we can utilize cache space more efficiently. Note that Figure 3.1 shows the ideal case of perfect profiling, which is likely to be hard to implement in practice. In this chapter we focus on this ideal case to highlight the potential benefit of grouping users and leave the exact grouping methods for future work. More specifically, we make the following contributions: 1. We propose a novel profiling strategy, where requests are grouped according to interest similarity. 2. Cache servers receive similar requests with higher probability. This increases cache hit rates and reduces redundant copies in caches. 3. After grouping, cached popular content can serve more users which saves cache resources and additional resources could be offered to non-mainstream users.

In this chapter, we only evaluate the improvement of cache performance brought by grouping. Defining the practical grouping mechanisms is left

for future work. The rest of the chapter is organized as follows. Section 3.3 introduces related work. Section 3.4 describes the network architecture. Section 3.5 presents the simulation model. Section 3.6 presents the results and discusses their impact. Finally, Section 3.7 concludes the chapter and presents directions for future work.

## 3.3 Related Work

Researchers have proposed some work related with edge computing. Some work such as [18] also provides ideas regarding cloud computing resource provisioning. Work in [23] and [24] put forward elastic or dynamic resource scaling schemes. However, profiling edge computing resources based on user interest similarity is still an open issue. Although researchers have proposed user interest recognition for some years, most of related work still remain on the level of identifying individual users. For instance, [25] introduces identifying users' preferences based on click history. Work in [26] proposes an aggregation of user profiles from multiple domains on social web. Some work has already addressed large scale user interest analysis such as [20]. However, that work is more about relation between overall user behavior and popularity of videos. On the other hand, some work also propose analyzing user interest such as [27], which put forward to merge user profile for better recommendation system. Authors in [28] characterize user viewing behavior and use Mixed Integer Programming to place the segments of videos optimally. A more similar work was proposed in [29]. However, the authors use one abstract function to represent the user interest similarity. Similarly, [30] indicates user interest similarity with a variable. The most important difference between these and our work in this chapter is that *we evaluate several concrete parameters of user interest similarity to better understand this problem.*

As to content delivery, CDNs such as Akamai use caching overlays to distribute content more efficiently [7]. The major feature of a CDN is distributed servers and cache hierarchy. Information-centric networking (ICN) proposes, among other aspects, to exploit in-network caching to enable more efficient content distribution by addressing content via a unique name.. Some work has been done focusing on reducing in-network caching redundancy and improving caching efficiency in ICN. ProbCache approximates the caching capacity of a path [31] . Guo et al. proposed a collaborative caching guided by content popularity rank [32]. In [33], the authors evaluate the performance of in-network caching and conclude that content popularity is one of the most important parameters.

As outlined above, Most of related proposals focus on utilizing caches based on content popularity. There is not much work referring to leveraging grouping users according to interest similarity for better cache performance in CDN, ICN, or other networks. Currently, we believe that the following, still unanswered, questions need attention:

1. Would grouping users provide benefits? In this chapter, we show that (in the ideal situation) grouping users improves overall cache hit ratio markedly.

2. How to implement user grouping, including pattern recognition and clustering? We need find out the optimistic algorithm for recognizing user interest patterns and clustering them.

3. How to optimize cache deployment and redirection of user requests? How to balance of the benefit of grouping users and the latency brought by redirection.

In this chapter, we focus on identifying the *ideal benefit* of grouping users regarding cache hit ratio, i.e., the first of the three open questions above; others are left for future work. We consider situations of different user interest distribution and evaluate the difference in cache hit ratio. This chapter is the first to consider explicitly the influence of different user interest profiles and how they impact caches near the edge of the network. Previous works have mainly focused on optimizing the performance of the cache but have not considered re-direction of users based on their interest profiles.

## 3.4 Network Topology

In this chapter we use various network topologies to study the effects of edge cache deployment and user grouping on the edge caches. Edge cache deployment makes it easier to understand the different cache performance in situations with different workload and grouping methods. It also helps isolate the benefit of grouping user from other scheme such as in-network caching etc.

We use a self-defined topology "Datacenter" and several real topologies including "Tiscali", "Garr" and "Geant". "Datacenter" is a two-layer network which has one core node serving as origin content source, 16 edge routers serving as edge cache servers and 10 users connecting to each edge server. Garr and Geant are parsed from the Internet Topology Zoo dataset [34] and Tiscali is parsed from Rocketfuel dataset [35]. Some

Table 3.1: Network topologies used in the study.

| Topology | Source nodes | Routers | Receivers |
|---|---|---|---|
| Tiscali | 44 | 160 | 1636 |
| Garr | 13 | 27 | 291 |
| Geant | 13 | 32 | 328 |
| Datacenter | 1 | 16 | 160 |

adaption is made to the topologies according to the assignment of simulation. For instance, we add 10 users to each edge servers.

The adaption helps us focus on the influence of grouping users on cache performance regardless of other factors such as different number of users connecting to cache servers. The properties of the topologies are summarized in Table 3.1. The cache servers are chosen based the centrality of the routers.

## 3.5 Simulation

We used Icarus as the simulator for evaluating the performance of grouping. Icarus is an ICN simulator for evaluating cache performance [36]. For the user interest distribution function, we use the widely accepted Zipfian distribution [37] as follows:

$$f(k;\alpha,N) = \frac{\frac{1}{k^\alpha}}{\sum_{n=1}^{N}\frac{1}{n^\alpha}} \tag{3.1}$$

where frequency of content k out of a population of N contents is $f(k;\alpha,N)$. N is the number of overall contents and k is the ranking of the content. We also use k as the ID of content here. Our simulation concerns several parameters as follows:

- $\alpha$ is the value of the exponent characterizing the interest distribution. In the simulations, we set $\alpha$ as 0.8,1.0,1.2.
- *rank* indicates the integer interval of content IDs that each user requests for. In the simulations, we set the interval as a population of $3 * 10^5$ content.
- *rank_similarity* indicates the overlap ratio of different ranks. In the simulations, we set the default

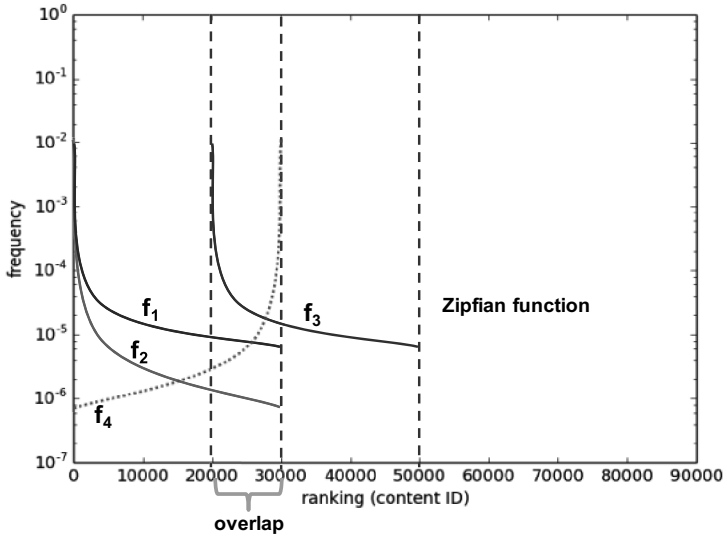



Figure 3.2: Example interest distributions.

*rank_similarity* as 0, so that users with different *rank* would have entirely different requests.

- *rank number* indicates the number of different popularity rankings. In the simulations, we set the default *rank_num* as 7, so that workloads before grouping have 7 different ranks of interest distribution. In other words, there are by default 7 different types of user profiles in the simulation.

- *cache_size* indicates the total size of network cache as a fraction of content population [36]. The default *cache_size* is set to 0.001.

- *workload* indicates the user request distributions with different grouping schemes.

Figure 3.2 shows an example of how the different ranks and rank similarities are implemented. We show 4 example interest distributions, f1–f4. The x-axis shows the content items ranked according to their popularity (possibly different for each interest distribution). The y-axis shows the number of requests generated by the interest distribution for each content item. Since there are a total of 4 different popularity rankings, the *ranknumber* is 4. Curves f1 and f2 share the same popularity ranking and their *rank_similarity* is 1 since they cover the exact same range of content.

However, they have different $\alpha$ values since the actual popularities of the objects are different. Likewise, f4 has a *rank_similarity* of 1 with both f1 and f2, but it counts as a different *rank* since the popularity of the objects is inverted. Requests from f3 overlap with f1, f2, and f4 by 33.3% so their *rank_similarity* would be 0.33. Since the shapes of f1 and f3 are identical, they share the same $\alpha$. Using this kind of a model as a basis, we generate the various workloads used in the simulation.

Our goal is to investigate the effects of the various parameters ($\alpha$, *rank_similarity*, and *rank number*) on the performance of the caches in the network. As a first step we focus on evaluating only cache hit rate, but other caching metrics, e.g., byte hit rate, latency, could also be used. The above model is flexible enough to support a variety of scenarios. The workloads we consider are as follows:

- **R**: Users have interest distributions with different *rank* but same $\alpha$. *rank_similarity* is 0 which means user groups have entirely different interests. The number of groups is determined by the parameter *rank* but users are not grouped in any way.
- **R-GR**: Grouping users with same *rank* in **R** workload After grouping, the users connecting to same cache servers would have same interest distribution.
- **AR**: Users have interest distributions with different *rank* and different $\alpha$, but no grouping of users is performed.
- **AR-GR**: Grouping users with same *rank* in **AR** workload, but a group may contain users with different values of $\alpha$.
- **AR-GAR**: Grouping users with same *rank* and same $\alpha$ in **AR** workload, i.e., each group has a specific *rank* and $\alpha$.

The default *rank_similarity* is set to 0. We also run simulations with different *rank_similarity* to identify the effects of it. Most of the simulations have different *rank number* which is at most 7. The *cache_size* is set as [0.001, 0.003, 0.005, 0.007, 0.009, 0.01]. The number of users connecting to each cache server does not change after grouping.

Table 3.2 shows the detail of the workloads. The word "locally" indicates the interest distributions of the users connecting to same cache servers. The different simulations aim at identifying the benefit of grouping users regarding cache hit ratio in different scenarios. For instance, in **R** and **AR** workload, we tried different numbers of *rank*. It can help us identify the factors influencing the benefit of grouping users. Since we keep

Table 3.2: Workload definitions.

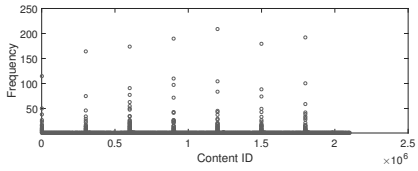
| *Workload* | *rank number* | $Num\_of\_\alpha$ | $Num\_of\_distribution$ |
|---|---|---|---|
| **R** | 1,3,5 or 7 | 1 | 1,3,5 or 7 |
| **R-GR** | 1(locally) | 1(locally) | 1(locally) |
| **AR** | 1,3,5 or 7 | 6 | 6,18,30 or 42 |
| **AR-GR** | 1(locally) | 6 | 6 |
| **AR-GAR** | 1(locally) | 1(locally) | 1(locally) |

each cache node serving the same number of users as before grouping, there are still some cache servers connecting to users with different interest after grouping. We think this is more close to reality due to the capacity of cache servers because it is hard to actually put all users with same interest under a same cache server and this reflects some inaccuracies in the grouping.

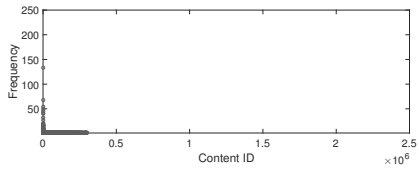
## 3.6 Results

The results are shown in Figure 3.3 separately for each of the workloads. We show one figure per scenario (**R**, **R-GR**, **AR**, **AR-GR**, **AR-GAR**) and each figure shows the requests observed by *a single cache* in the scenario. The x-axis shows the content IDs in the whole workload and the y-axis shows the number of requests received by that cache in the simulation. As we can see, grouping users helps significantly narrow down the range of requests that a cache sees. For instance, in Figures 3.3a and 3.3b we clearly see the effects of various popularity rankings in Figure 3.3a as spikes in requests. When users are grouped according to their interests, the request pattern is much close to a single zipf-distribution, as shown in Figure 3.3b. Likewise, Figure 3.3e has a more concentrated request distribution than Figure 3.3d and Figure 3.3c, as expected. Given that we use *a priori* knowledge to assign groups perfectly, the results reflect an ideal situation, but they clearly illustrate the potential of this method.

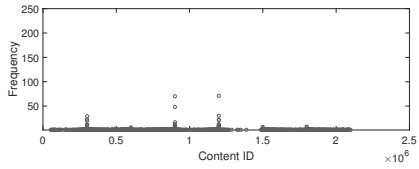
The above results show the effect of grouping on the traffic workloads of the caches, and we now turn to evaluating the performance of the cache as a function of the various parameters described in Table 3.2. Although the figures below show results for all workloads in the same figure, the exact numbers can only be compared among (**R**, **R-GR**) and (**AR**, **AR-GR**,



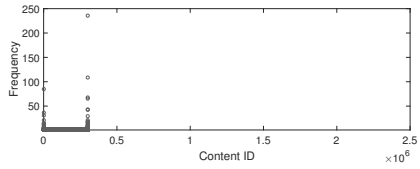

(a) Request distribution with workload **R**, in which there are 1 $\alpha$ and 7 *rank* as per Table 3.2.



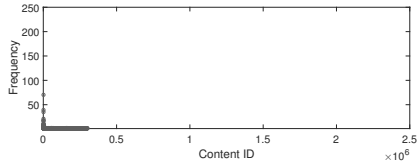

(b) Request distribution with workload **R-GR**, which is the request distribution for a cache server after grouping Figure 3.3a. There is 1 $\alpha$ and 1 *rank* as per Table 3.2.




(c) Request distribution with workload **AR**, in which there are 7 $\alpha$ and 7 *rank* as per Table 3.2.




(d) Request distribution with workload **AR-GR**, which is the request distribution for a cache server after grouping Figure 3.3c according to *rank*. There are 7 $\alpha$ and 1 *rank* as per Table 3.2.




(e) Request distribution with workload **AR-GAR**,which is the request distribution for a cache server after grouping Figure 3.3c according to both $\alpha$ and *rank*. There is 1 $\alpha$ and 1 *rank* as per Table 3.2.

Figure 3.3: Request distribution in different workload.

and **AR-GAR**) because of the different $\alpha$ values in the experiments. The exact hit rate numbers are not as important as the relative differences in performance in the different workloads.

The key findings can be summarized as follows:

1. *cache_size*: As Figure 3.4 shows, the cache hit rate increase with bigger cache size, as is to be expected. As we see, **AR-GR** and **AR-GAR** improve the cache hit rate compared with **AR**; **R-GR** likewise improves over **R**. This result shows that grouping users can benefit cache hit rate independent of cache size.

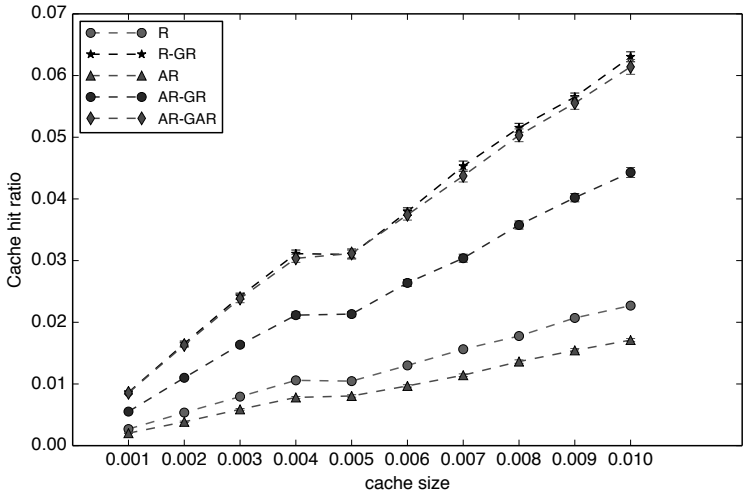



Figure 3.4: Cache hit rate vs. cache size.

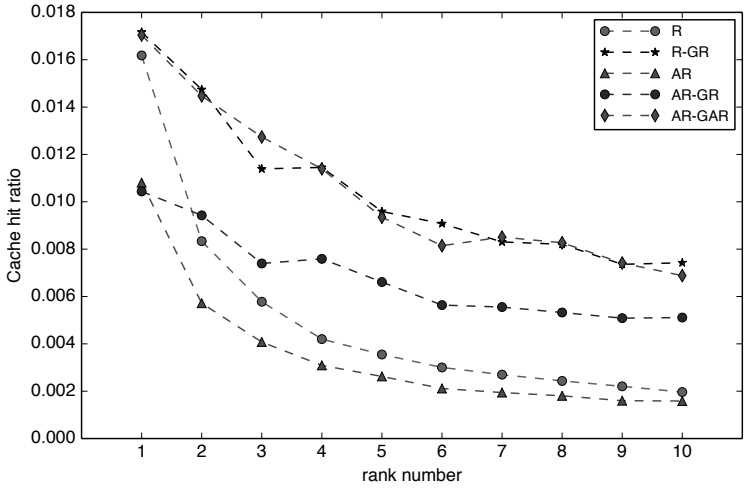



Figure 3.5: Cache hit ratio vs rank number.

2. *rank number*: As Figure 3.5 shows, cache hit rates of all workloads decrease when *rank number* increases. This is expected since more ranks (i.e., more sets of interest distributions) will bring more different request distributions. However, grouping serves to alleviate the effect and remains effective even with a large number of different ranks. Note that the cache size used in Figure 3.5 is 0.001, i.e., the smallest in our study. Larger cache sizes exhibited the same kind of behavior.

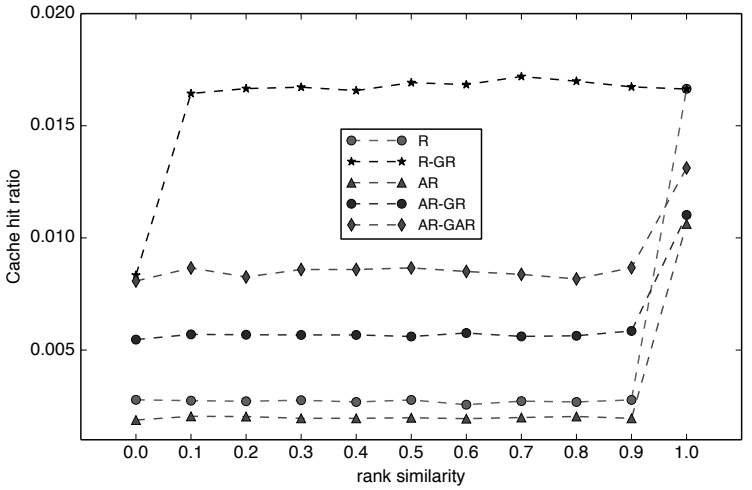



Figure 3.6: Cache hit ratio vs rank similarity.

3. *rank_similarity*: As mentioned before, the default *rank_similarity* is set to 0 in most simulations. We also investigated the effects of overlapping interests by varying the similarity from 0% to 100%. As Figure 3.6 shows, *rank_similarity* does not influence cache hit ratio much for most of the cases. Two notable observations are evident in the figure. First, all except **R-GR** show a jump at the end when similarity reaches 100%. The explanation is that when similarity reaches 100%, the distributions become identical (except for different $\alpha$) and therefore all caches see the same traffic. Another is the jump in **R-GR** when similarity increases beyond 0%. While we do not have a full explanation for this phenomenon, we conjecture the reason to be related to way we have implemented similarity by sliding the request distributions over one another, as Figure 3.2 shows.[1] Understanding the reasons behind this and investigating it thoroughly are left for future work.

## 3.7 Conclusion and Future Work

In this chapter we have considered the problem of grouping users for more efficient request processing in edge caching scenarios. Our simulations show that grouping users can improve cache hit rate in many different scenarios. We have investigated various grouping strategies and results have consis-

[1] We have verified the simulation code and ruled out errors in there.

tently shown that when user interest profiles are different from each other, grouping users of one interest profile into one edge cache yields considerable benefits in terms of overall cache performance. Although our work was done under the assumption of an ideal distribution of user groups to caches, it highlights the potential for improvement in caching by this simple technique.

Future work needs to tackle three main issues. First, we have assumed that any user can be re-directed to any cache, regardless of the locations of the two. While in principle this is feasible to do, it may result in significant increases in user-perceived latency. A more realistic look at which clients could be re-directed to which caches could be included as part of the work. Second, it is unlikely that the number of user groups is smaller than the number of edge caches, thus one cache may be forced to serve multiple user groups with different interest profiles. An obvious solution would be to attempt to assign user groups that have similar interest profiles in the same caches. Third, we have not considered the practical implementations of how the groups are formed. Various existing classification and clustering algorithms can most likely be used to achieve the groupings and evaluating their efficiency is part of our future work.

# Chapter 4

# Resource Grouping

## 4.1 Overview

Edge clouds handle data and computations closer to its source and users. Applications like industrial automation, bring new challenges and require solutions tailored for computation-centric edge cloud networks. In this chapter we build on existing edge and fog computing models and develop a solution to predict and store data in edge resource caches for upcoming computations. Our solution is based on grouping caches according to the workloads they serve. We further develop methods for populating the caches and ensuring the coherence of the cached data. We evaluate the performance of our grouping mechanisms and show that they bring significant performance gains, both in terms of network traffic and access latency.

## 4.2 Introduction

Edge clouds are a new and attractive way of handling large-scale data analysis closer to the clients at the network edge. Edge clouds offer several benefits including decreased latency for clients, reduced network traffic, and better handling of information that is of local interest. Different models for edge computing have been proposed [38,39] and we follow our proposed Edge-Fog cloud model for this work [39]. *Edge-Fog cloud follows a three-tier hierarchy which consists of lower-powered edge devices closest to the users, fog devices with more computational power, and a central data store for permanent archival of data.*

While the data store provides permanence, solely relying on it for storing computational data adds considerable delay for fetching data to edge resources. Hence, caching data at the edge seems to be the obvious answer

as it yields several benefits [40, 41] as well. However, we need to address additional challenges on how to manage, discover, and use the data cached at the edge. Existing solutions [40] propose CDN-like models which is not appropriate for a computation-first network as necessary for edge cloud application scenarios. When compared to CDNs, data in Edge-Fog cloud has shorter temporal relevance and receives more frequent updates.

In this chapter, we propose an efficient edge caching mechanism leveraging the edge and fog resource caches to predict and store data required for upcoming computations. Our target applications are in industrial environment, particularly in factory automation and collaborative robotics. This chapter makes the following contributions. First, we define a model and methods for cache grouping in an Edge-Fog cloud. Second, we develop mechanisms for ensuring coherency of cached data. Third, we evaluate the performance of our grouping solutions in a simulated Edge-Fog environment and show that grouping based on workload type brings significant performance gains such as reduced network traffic and access latency. We also discuss the optimal size of such resource groups and importance of workloads in the system.

## 4.3 Application Scenarios

Recent studies predict close to 50.1 billion IoT devices will be connected over the Internet by 2020 [42]. Data generated by these devices will require time-critical processing and management to support fault resistant applications such as augmented reality, autonomous driving, video analytics etc.

Automation also extends to factories and acts as a driving force behind next generation manufacturing industries. The production system needs to be made faster, flexible, and cost-efficient to cope with increasing demands. Factories can achieve low latency computations by allocating tasks on edge clouds. Edge nodes can process data from automated tools and sensors to be reconfigured based on the task requirements.

However, factory tasks rely heavily on the availability of required data at compute time. Specification of end products can significantly vary, requiring on-the-fly calibration of the tools for each workpiece. Such recalibration information must be cached at edge nodes to ease subsequent task processing. For example, a machine meant for drilling holes must be able to change its settings for the next workpiece or switch to a different task altogether such as driving screws. Further, industries have started collaborative robots such as Bosch APAS [43] that work in tandem with human operators. Such robots need time-critical processing to create a safety zone

for its operator while executing future demands. Relevant warning and sensor information must be cached at edge nodes to achieve sufficiently low processing requirements.

Autonomous transportation systems within a factory also impose similar requirements by requiring optimal and updated routes in extremely low time bounds. Required map data must be pre-cached and updated to all vehicles with critical information such as accidents or path congestion within a reasonable time. Augmented Reality glasses can assist operators in a continuously varying production environment by performing markerless object recognition and accurate tracking in a factory. This requires comparing real-world objects with pre-created 3D models stored in a remote data store. The fluidity and QoE of these devices significantly rely on bounding data retrieval delay within human reaction time.

Apart from the applications mentioned above, many other Cyber Physical Production Systems data in edge clouds needs inter-operation and communication which can only be achieved by efficient caching and data sharing within cloud resources.

## 4.4 Related Work

Several edge cloud models such as Cloudlets [44], nano data centers [45,46], community clouds [47], CISCO Fog [38] have been proposed to perform computation tasks at the edge of the network. However, these models assume that the required data for computation at a node is available in local caches of edge resources. That does not always hold as edge cache hit ratio is heavily dependent on the deployed workload type [7]. Further, they do not consider the impact of fetching the data from a data center into the edge cache and the subsequently added delay on workload computation.

Content Delivery Network (CDN) models aim to distribute content to end users via distributed servers and edge cache hierarchies [48, 49]. Edge caching is also a major motivation behind the design of 5G technology [50]. Exploitation of in-network caching to enable more efficient content distribution serves as a motivation behind information-centric networking (ICN) research.

However, both CDN and ICN assume a single publisher/owner of the data which holds the rights for updating that content in future [41]. These networks follow a *push-based* approach wherein the owner pushes its data update to the central repository which broadcasts that update to every edge cache hosting that data. However, such an approach is inefficient for computational edge caches as the local copy of shared data can frequently

be updated simultaneously by several edge resources. Notifying the central database of every update can lead to a severe network congestion and does not scale.

Cooperative caches groups at the edge of the network have been proposed to avoid recurrent updates to the central database. Ramaswamy et al. [51] clustered edge caches into cooperative groups based on their proximity to other caches and the origin server. Our proposed cache grouping technique significantly differs from their approach. The authors cluster resources based on their network distances to other resources whereas we consider locally cached content as clustering classifier. Using network distance for clustering in Edge-Fog cloud would significantly lower the efficiency of task deployments which also considers the processing power of Edge/Fog resources. Furthermore, Ramaswamy et al. they do not consider parallel updates within cache groups and therefore do not propose a coherence model to mitigate invalid simultaneous updates.

## 4.5 Resource Cache Grouping

Resources in Edge-Fog cloud request data from data store into their local cache according to end application requirements. Task deployment algorithms for Edge-Fog cloud, such as LPCF, designates a set of resources (Edge and Fog alike) for available tasks on to achieve least processing and network cost involved [39]. However, in a system with varying workloads, such a deployment reduces cache re-usability as same set of resources might be allocated to workloads with different data requirements in subsequent computations. This further leads to higher cache misses and higher network latency for fetching required data from data store into local cache thereby delaying the overall computation.

We consider Edge-Fog compute resources as collection of edge caches represented by $RC = RC_0, RC_1, ..., RC_{n-1}$. Every edge cache stores data $D_i$ as per its application task requirement. Deployed tasks in cloud can be classified into workloads $W$ of $k$ types. A resource computing workload $W_k$ will require data classified according to that workload $D_i^k$ in their cache. We propose a cache grouping algorithm which aims to cluster caches into cache groups $CG = CG_0, CG_1, .., CG_{K-1}$ based on their local cached content classification. Cache groups are not disjoint as resource $RC_x$ can be part of more than one $\{CG\}$ if it has cached data for different workloads. The size of the group, $|CG|$ denotes the number of members of that group. The caches within a group can maximize their cache hit ratios and lower network delays by sharing data with other group members in future deployments.

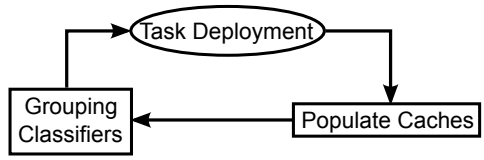



Figure 4.1: Edge-Fog cloud caching algorithm.

### 4.5.1 Grouping Algorithm

We propose a three-step iterative cache grouping algorithm which builds up on the available task deployment algorithms. The algorithm is shown in Figure 4.1. At time $t = 0$, none of the resources in Edge-Fog cloud have any tasks assigned to them and thus have no data in their local cache. At computation arrival time $t_c$, task deployment algorithm deploys an application task on a set of $\{RC\}_i$ resources which then retrieve the required data from data store into their local caches (phase "Populate"). As all resources involved in the computation belong to same task, they cache same or related content in their local cache. The computation is classified as part of workload $W$ and all resources $\{RC\}_i$ are grouped in a single abstract cluster $\{CG\}_i$. As several parallel computations are deployed on the cloud, at time $t = n$ computations deployed are classified in $W_k$ workloads which form $CG_k$ resource cache groups.

In the next iteration, the task deployment algorithm prioritizes deploying next application task on a cache group which handled that workload in the previous cycle. This enhances the cache re-usability in resources belonging to that group. In case the resources in a group are non-ideal for a task deployment, other resources (independent or other group members) are considered for computation. Size of a cache group increases as more resources compute tasks and cache content for that particular workload. As a result, a cache can be member of more than one group based on its cached content classification. Figure 4.2 shows a snapshot of Edge-Fog cloud resources which have been grouped in two cache groups.

Within each cache group a resource is assigned as a *leader* (depicted with crown in the figure) which acts as a representative and communication backbone of the group. The leader is responsible for maintaining a coherent copy of data within a group and to enable content sharing among group members. The leader is required to have a consistent connection with all of its group members, exploiting the Fog resources in the model if needed. A distributed election algorithm can be used to elect a group member as a leader. The group leader can also replicate its local data structures on a secondary node which acts as a backup leader to ensure consistency despite failures.

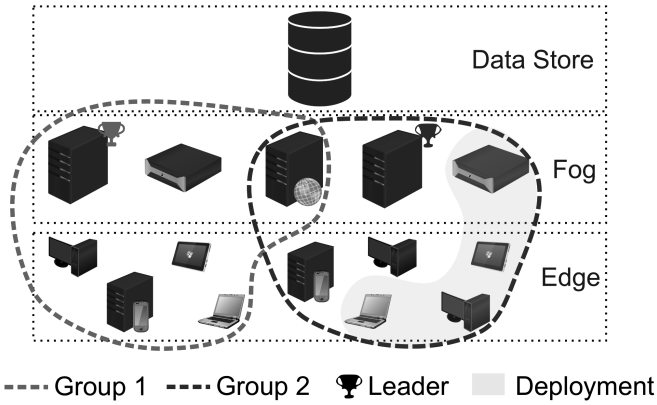



Figure 4.2: Edge-Fog cloud grouped resources.

### 4.5.2 Grouping Classifier

Our cache grouping algorithm relies on data classification in the Edge-Fog cloud. As mentioned in Section 4.3, different applications require different sets of data at different times for their compute tasks. Data can be clustered according to their similarities which can be exploited to form dedicated cache groups for a data type. Several classification metrics can be used to achieve such groups.

1. **Location**: Data can be classified according to its location of generation or usage. For example, Augmented Reality headsets require 3D model and augmentation based computations only on objects within their field of vision.

2. **Relevance**: Data sharing attributes for re-configuration is a good classifier. Collaborative robots submitting production tasks of mobile and laptop cases can be grouped under `casing` attribute.

3. **Pending tasks**: Factory environments are flexible and dynamic where tasks are not bound to robots; instead the robots choose from a pool of pending tasks.

4. **Time**: Data generated by sensors are relevant to the end resources only for a particular period which can range from a few hours to a couple of days.

5. **Personalized settings**: Collaborative robots work with human operators who can configure them according to their specific needs.

6. **Routes/maps**: Autonomous robots are often mobile and require constant computation of optimal paths devoid of hinderances. Continually updated maps must be made available at frequent time intervals.

7. **Warning signals**: Data relevant to the safety have higher importance over other information and need to be made available to all the devices until categorized as invalid.

8. **Sensor information**: Actuators respond and regulate themselves based only on particular sensor data.

Above are examples of possible classifiers relevant for a factory automation scenario. However, for optimal operation, one or more groups need to be considered at the same time and groups must be weighted differently for each category of end devices, as not all the information is equally relevant.

Resources clustered in a cache group need to communicate with their group members to share updated data efficiently. However, an effective communication technique for efficient operation needs to fulfill the objectives as follows.

1. Reduce *unnecessary network traffic* by exchanging data between resources only when needed.

2. Ensure *consistent copies* of data by avoiding computation on stale copies.

Considering above objectives, we propose a communication model which introduces a set of tabular data structures attached to resource's cache. We further present a low overhead message flow model to update and retrieve shared data within a group. Our model ensures *causal coherence* on shared data and is highly inspired by *directory cache coherence algorithm* [52] for networked processing systems.

### 4.5.3 Cache Data Structures

We now define data structures deployed with resource caches to assist data sharing within the group. The data structures provide content information to data stored in resource local cache. We deploy tables at three entities in the system: member resource, group leader and the Data Store.

**Group Member Table:** Every resource in Edge-Fog cloud associates itself with a content group and maintains a table to help content sharing with other group members. Members maintain a local cache table with the following entries:

1. **Data Name**: URI of a content cached in local cache.
2. **Leader Address** Address of the group leader responsible for synchronization of that content.
3. **Tag** State of data within its local cache. If data is currently used for computation, tag entry is *locked* otherwise *free*.

The group member table maps locally cached data to groups based on their respective *leader address*. *Tag* is required for providing coherence within the group and is explained in further sections of this chapter.

**Group Leader Table:** The group leader acts as a communication gateway between members of the group. To maintain the current state of data flowing within the group, the leader maintains a group leader table with following information:

1. **Data Name**: URI of the content cached within the group resource caches.
2. **Tag**: Maps to content tag in member resource cache.
3. **Resource Address**: Address of resource which updated the data. The resource also acts as host of that content within a group.
4. **Timestamp**: Time at which resource notified the leader after updating the content.

The *group leader table* helps ensure that the leader has addresses of host resources and the content state is synchronized between a resource and group leader.

**Data Store Table:** Data Store is the central repository and backup of cached content in Edge-Fog cloud. Resources update their content in the Data Store after every computation. The Data Store table has the following entries:

1. **Data Name**: URI of data stored in Data Store.
2. **Classifier Type**: Classification property used for mapping content to a cache group.
3. **Leader Address**: Address of group leader handling synchronization of that content.

### 4.5.4 Communication Flow

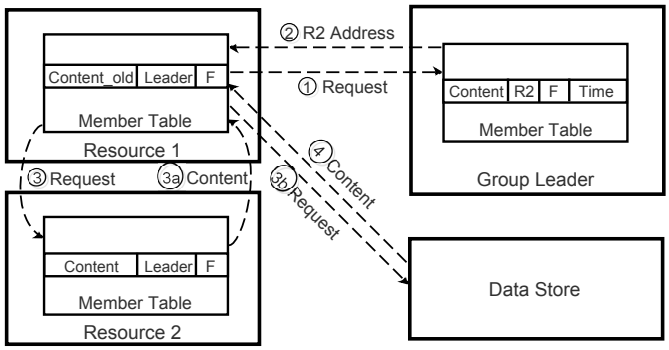
We use the information stored in data structures described in previous section for ensuring that content gets updated properly. We use a *pull-based* model within a group which limits the number of messages in the network while ensuring data consistency. We assume that the data store acts as a central roll-back in case of failures in the system. To ensure this, the resources upload their updated data to the Data Store after each successful computation. As data upload happens *in-parallel* to task computation and data retrieval, it does not impact the computation time in the cloud. We further assume that messages in the system are not lost or corrupted in transmission.

**Retrieving Content:** A naive way to update cached content is to retrieve it from the data store. However, the main objective of forming cache groups in Edge-Fog is to assist content sharing amongst the computing resources. Retrieving content within a cache group must also preserve the coherence among multiple content copies in other edges of the network.

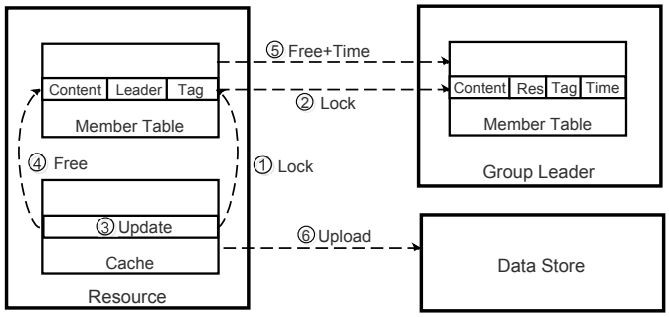
The communication model for retrieving content within a group is shown in Figure 4.3a. The model ensures *causal coherence* by sharing only last known updated content within the group. The group leader acts as information dissemination entity for the group. Every resource requests updated copy of content before initiating computation on its locally cached copy. The request is sent to the group leader which checks its table and returns the address of the node which last updated the content. The requesting node directly queries for the content from last-updater node. In case the node is alive and has the data in its local cache, it sends the content back to the requester. Otherwise, request is sent to the data store to retrieve backup copy.

**Updating content within a group:** Content in an Edge-Fog cloud resource group is continuously updated after each successful computation. However, to mitigate invalid results, resources must always compute on the most relevant copy of required data. A naive approach is to push the updated data to all members of the group which house copy of that data in their local cache. However, this leads to unnecessary flooding in the network which impacts network latency.

Instead, we employ a pull-based, step-wise checkpoint approach for handling updates. The message flow is shown in Figure 4.3b. Before computing on a locally cached copy of content, a member resource inquires its group leader for any updates on that copy of data. In case data has been updated by another member, the requesting node retrieves the latest copy by following the model described above. After successful retrieval, the updating




(a) Data retrieval within cache group.




(b) Data update within a cache group.

Figure 4.3: Communication Model.

node marks its *"update-in-progress"* by tagging its locally cached content as *locked* and asks the group leader to do the same. After a successful update, the resource un-tags its cached content as *free* and notifies the leader of the completed update. The leader, in turn, marks the particular content as *valid* along with the timestamp of the operation. This operation prevents any other resource to retrieve data under update. Finally, the resource updates the Data Store with the computed data.

## 4.6 Evaluation

We implemented our system in Icarus [36] on a topology of 320 Edge and Fog resources and a central data store which stores all content in network. We clustered resource caches into evenly divided groups of various sizes. A Fog resource in each group is assigned the task of group leader.Network delays were modeled according to [39,53]. A workload is defined as a request distribution following a power law distribution. We generate requests for

$96 * 10^4$ content items divided in upto 32 different workloads. A resource can store maximum of 10% of overall contents in the network in their local cache. Caches utilize Least Recently Used (LRU) cache replacement policy for swapping their cached contents. Cache retrieval and updates follow the communication models described in Section 4.5.4 coupled with ideal Nearest Replica Routing (iNRR) algorithm [54].

### 4.6.1 Grouped vs. Non-Grouped

We first compare the effect of grouping on system performance. Figure 4.4 shows the cache hit rate and network latency after grouping resource caches. For optimal comparison, we cluster caches into same number of groups as the number of workloads deployed to ensure 1:1 mapping (see below).

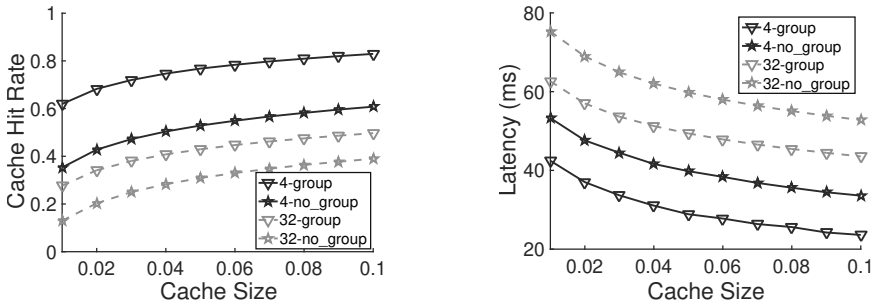
Figure 4.4a shows cache hit rate after grouping. For both 4 and 32 workloads, grouping almost *doubles* the overall cache hit rate. Similarly, Figure 4.4b shows that the latency of fetching the content decreases by up to 45% after grouping. The results clearly indicate that our grouping strategy significantly improves content management in edge clouds.

**Effect of Cache Size:** The results also show that cache grouping is most effective when cache sizes are small. As we increase the cache size of computing resources, the cache hit and latency gains slightly diminish. The reason behind this decrease is the overall fraction of the content that can be cached in the system. When resource cache has limited size, the amount of content that it can cache is low and grouping several resources in cache groups increases the probability of serving locally cached content. On the other hand, as cache size increases, it can cache more content in the network which increases cache hits even in disparately placed resources, closing the gap between non-grouped caching and grouped strategies.

**Effect of workload sizes:** Figure 4.4 also shows a correlation between performance gain and workload size. Both cache hit ratio and latency performs much better for lower workload sizes. *A workload is modeled as uniform distribution of similar content requests.* Lower workload sizes depict that overall content requests are very similar and thus can be satisfied by local caches of resources. As the content requests start being more unique, resources undergo more cache misses thereby inducing latency while satisfying a request.

### 4.6.2 Variable Group Size Analysis

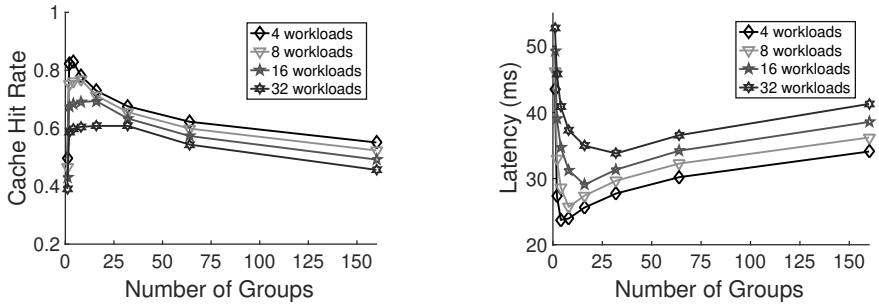
We analyze the effect of group size $|CG|$ and number of groups $\{CG\}_k$ on system performance. Figure 4.5a shows the impact of variable group




(a) Effect on cache hit rate.

(b) Effect on latency.

Figure 4.4: Variable grouped resource cache size analysis.




(a) Effect of group sizes on cache hits.

(b) Effect of group sizes on latency.

Figure 4.5: Resource cache grouping analysis for various workload sizes.

sizes on cache hits in the system. We compare the performance for several workload sizes and analyze how they affect the cache performance. For all workload sizes, cache hit rate is is observed to be low as due to lack of cache grouping, content requests are handled by all resources in the cloud. As resources grouping in the system increases, the request types start converging on a single cache group. Maximum hit rates are achieved when *number of cache groups equal the number of workloads*. This 1:1 mapping ensures that each workload is being handled by a dedicated cache group, eliminating any overlap. Increasing the number of cache groups more than available workloads leads to overlap and duplication of content which reduces the cache hit performance of the network. A similar trend is also seen for latency in Figure 4.5b. The content retrieval latency is inversely proportional to cache hit rate of the system. As the cache hit rate increase, the latency to retrieve content decreases and reaches a global minimum at 1:1 deployment of workload and groups.

## 4.7 Conclusion

We have presented a grouping strategy for managing content in edge cloud caches. Our grouping is based on classifying content based on their workloads and caching related content on same caches. Our communication model provides causal data coherence while enabling parallel updates. We have evaluated our approach via simulations and have shown that grouping based on workload type significantly improves system performance in edge clouds through reduced network traffic and access latency. Our results show that the optimal number of groups is the number of workloads on the system.

# Chapter 5

# Augmented Vehicular Vision: Potential Exploration

## 5.1 Overview

Vehicular communication applications, be it for driver-assisting augmented reality systems or fully driverless vehicles, require an efficient communication infrastructure for timely information delivery. Centralized, cloud-based infrastructures present latencies too high to satisfy the requirements of emergency information processing and transmission. In this chapter, we present a novel Vehicle-to-Edge (ARVE) infrastructure, with computational units co-located with the base stations and aggregation points. Embedding computation at the edge of the network allows to reduce the overall latency compared to vehicle-to-cloud and significantly trim the complexity of vehicle-to-vehicle communication. To demonstrate the efficiency of our solution, we apply these principles on an augmented reality head-up display. In this use case, vehicular communication is exploited to connect vehicle's vision, and quickly propagate emergency information. ARVE is a general system framework, applicable to many practical scenarios. Our preliminary evaluation shows that ARVE noticeably decreases transmission latency with reasonable capital expenditure.

## 5.2 Introduction

Connected automated driving has recently become closer to being a reality. In 2018, California and Shanghai authorized the deployment of autonomous vehicles on public roads for testing purposes [55, 56]. Vehicular communication systems play a key role in sharing information between vehicles and

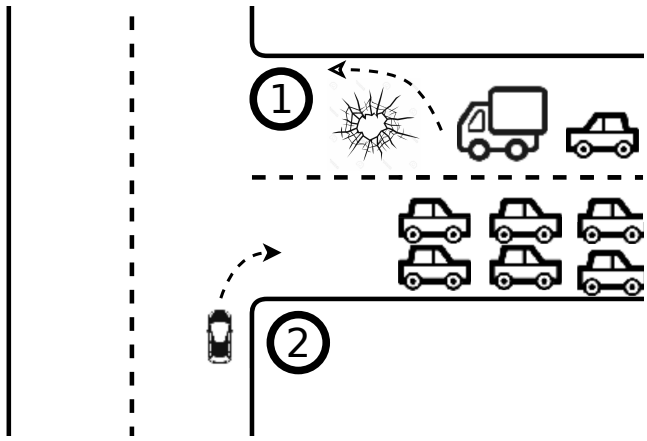



Figure 5.1: Common connected vehicles scenarios.

roadside infrastructure units (RSU) . Use cases include emergency warning system for vehicles, cooperative adaptive cruise control, collision warning etc. Current solutions focus on three types of communication: vehicle-to-vehicle (V2V), vehicle-to-cloud (V2C), and vehicle-to-roadside infrastructure (V2I) [57,58]. Although these solutions fulfill basic demands, *efficiently sharing complex and large volumes of data among vehicles at scale* remains a challenge.

Figure 5.1 illustrates two related scenarios: 1. The leading truck encounters an unexpected pothole. The truck notifies the following cars to avoid a potential accident. 2. Congested traffic is out of sight for cars planning to take the road on the right. Once aware, these cars will choose a better path. In V2C, even though the leading truck immediately uploads the captured pothole information, the combined latency of transmission, processing and distribution may be too high for the following vehicles to avoid it. Similarly, the connection establishment time of V2V communication with the complexity of forwarding information in a constantly varying crowd of nodes can lead to vehicles having only partial knowledge of the situation. V2I provides better data distribution; however, sharing accurate emergency information entails nontrivial computation and coordination. Roadside infrastructures should therefore integrate computing features for fast and reliable emergency information propagation.

Edge computing facilitates latency-sensitive workloads by performing data processing in Edge Servers (ESes) located close to the user. The gain in latency provided by edge computing can be considerable. In Table 5.1, we measured the round trip latency for various servers through an LTE

| Edge | Nearby Cloud | Far Cloud |
|---|---|---|
| 19.9 ms | 24.9 ms | 52.4 ms |

Table 5.1: Average network round-trip latency over LTE to different targets.

network: the first pingable IP, noted as *Edge*, a cloud server located in the same city and another server 1000 km away. Unsurprisingly, the latency to the closest server is half the round trip time to the furthest cloud server. Moreover, the ES presents a 20% improvement compared to the nearest cloud server, making it an attractive location for latency-sensitive applications.

We propose to *use vehicle-to-edge (V2E) to enhance vehicular communications*. Our design, ARVE, is a framework designed to be independent of the actual protocols, in order to allow it to apply equally to current as well as future networks. We choose to apply those principle to Connected Vehicle Views (CVV), a concrete use case of ARVE (see Section 5.4 for a detailed discussion of this use case).

Our contribution is threefold:

- We present the design of ARVE, which equips RSUs with computation and cache capacity.
- Concrete application of ARVE to CVV using Augmented Reality Head-up Display (ARHUD). This use case displays the advantages of ARVE while scaling the problem of vehicle vision from a network perspective.
- We present a preliminary evaluation to show that ARVE offers noticeable performance improvements with reasonable expenditure in infrastructure.

The rest of this chapter is structured as follows. After a review of related work in Section 5.3, we introduce the ARHUD use case in Section 5.4. We then describe the system design, implementation and communication process of ARVE in Section 5.5. The potential network protocols are discussed in Section 5.6. Preliminary evaluation results are given in Section 5.7. We conclude the chapter in Section 5.8.

## 5.3 Related Work

Emerging technologies enable various functions for autonomous vehicles but also bring new challenges. Different protocols and standards, i.e., Direct Short Range Communication (DSRC), Device to Device (D2D) and 5G, improve data transmission [59–61]. However, the large volumes of data will challenge current computation resource deployments and risk making them bottlenecks. Edge computing as a solution to bring computation close to user, has attracted attention, such as [62] which explores an integration of 5G, SDN, MEC and vehicular network. Uncoordinated strategies for edge service placement have been investigated in [63] and the results have shown that they work well for this problem. Paper [64] discusses the direction of utilizing Information Centric Network and MEC for connected vehicles. Meanwhile, the fundamental issues, i.e., architecture design, communication process, network protocols and implementation concerns are largely yet to be explored. Efforts on developing vehicular applications have achieved some results [65, 66], but without an improvement from system and networking point of view, those applications face difficulties to scale in realistic situation.

## 5.4 Use case: Connected ARHUD

One of the main concerns in the automotive world is safety. According to the 2015 National Motor Vehicle Crash Causation Survey by National Highway Traffic Safety Administration (NHTSA), 93% of crashes are attributed to drivers, of which, around 74% are due to erroneous recognition or decision [67]. However, autonomous vehicles can also get "confused" easily. For instance, GM Cruise autonomous cars sometimes slow down or stop if they see a bush on the roadside [68] and similar issues exist in lane changes. As such, the intervention of a human driver is still critical for safety. To assist the driver, vehicles are outfitted with sensors and display devices which provide valuable information to the driver, such as about environmental condition and the driver's driving habits. The most common display method is a heads-up-display (HUD).

In recent years, augmented-reality (AR) HUDs have attracted both academic and industrial attention [66, 69]. AR can embed 3-D views of the information into the rendering background on the HUD, enabling accurate obstacle recognition and emergency notification. Previous work has put effort on matching the embedded information with the real environment, cognitive usability, visibility, among others [70, 71]. However, connecting

vehicle views via ARHUD remains a challenge. Recently, the authors in [65] explored how to share vision between two vehicles. Although the work proposed solutions for basic view transformation, it is still not enough to connect vehicle vision at scale under realistic concerns of bandwidth, latency and computational resources.

**Challenges** are multiple: 1. A crowd-sourced map which is the combined 3D point cloud from the independent real-time views of the connected vehicles, acts as a reference coordinate system to localize incidents. This map is too voluminous for both real time generation and transmission, hence we need to develop additional mechanisms to address its proper generation and maintenance. 2. Proper network protocol stack needs to be explored. There are several protocols proposed and tested in vehicular network. An integrate protocol stack fit for different applications is still beingless. 3. Privacy and security concerns may arise in distributed V2V communications.

In this chapter, we design ARVE, a framework designed to enable Connected Vehicle Views (CVV) where nearby vehicles are able to share their views, assisted by the edge components, and form a more holistic view of their current situation. This enables fast distribution of critical information, i.e., obstacle detection, emergency report and collision notification. While we use CVV as an example, ARVE can serve any similar application, which requires computation and short latency.

## 5.5 System Design

In this section, we describe the ARVE design. First, we explain our system architecture and describe the major communication processes in the system. Then, we propose an implementation scheme and present how to apply it to CVV.

### 5.5.1 System Architecture

We now introduce the ARVE architecture model. It has three key elements: environment, vehicles and edge servers. Environment includes the background road network, roadside buildings, infrastructures and pedestrians, etc., while the others represent the computational elements in the system. Figure 5.2 depicts our system architecture in which ESes are distributed hierarchically in two tiers. Some are co-located with base stations[1], while

[1]Base station in this chapter refers to the entity at the edge of the fixed network, e.g., BTS, eNB and gNB etc.

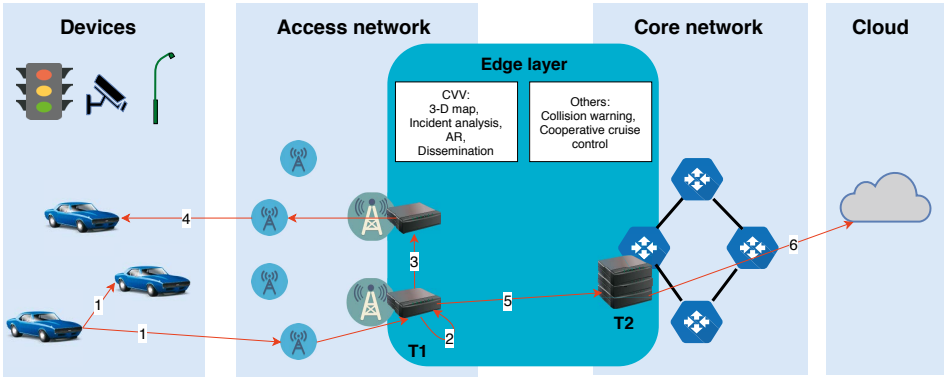



Figure 5.2: ARVE System Model. The numbers refer to the steps in the communication process (see Section 5.5.3).

others are co-located with aggregation points. We name the former Tier1 Edge Server (T1 ES) and the latter are Tier2 Edge Servers (T2 ES).

The edge layer is the amalgamation of T1 and T2 ESes and is where ARVE operates. Edge layer communicates with vehicles and RSUs via nearby radio access network, and transmits data with remote cloud for synchronization. Each T1 ES has a range over the area covered by its connected macrocell and surrounded small cells. The hierarchical design of the edge layer allows applications with different requirements to be processed differently for better performance. T2 ES collects data from multiple areas (multiple T1 ESes) to provide larger scale of service and data backup, e.g., to improve traffic flow by sending cruise control messages. T1 ES, which is closer to vehicles, serves applications with higher latency-sensitivity, e.g., emergency notifications.

### 5.5.2 ARVE Basic Operation

The basic operation of ARVE relies on the generation of a map around the vehicle, to enable awareness of the surroundings. The generation of the crowd-sourced map involves multiple steps. First. for each vehicle, we generate a 3D point cloud of the road in front of the vehicle using visual sensors present in the vehicle (e.g., LiDAR, RGBD camera). Then. the point clouds from multiple connected vehicles are transmitted to the edge server and combined into a 3D street view. Finally, the combined point cloud of the street is transmitted back to the vehicles, and each vehicle can display the street view according to its own position, so that the driver would be able to see the extended view of the whole street on the HUD.

### 5.5.3 Communication Process

Next we describe the six basic steps (marked in Figure 5.2) in ARVE: neighbor notification, data processing, transmission, dissemination, aggregation, and upload. The exact details depend on the actual application; here we use an emergency notification application to showcase the communication process:

1. **Neighbor notification:** The nearest vehicles require the fastest notification of emergencies. Therefore, upon emergency detection, a vehicle needs to warn its neighbor vehicles immediately, by sending simple notification via V2V. The notification includes only critical messages, e.g., name/type and coordinates of the emergency, to minimize V2V bandwidth usage and latency. The V2V notification is relayed until reaching a predefined maximum number of hops. Meanwhile, the vehicle sends a detailed report to nearest T1 ES via V2I. The report includes collected sensor and camera data with only the minimal, necessary data compression.

2. **Data processing:** Once a T1 ES receives a report, it processes the data and caches it for passing on to later passing vehicles. As discussed in [65], sharing views of incidents among vehicles is nontrivial. ESes maintain and update local map in real time, by collecting data from passing vehicles and synchronize it with a cloud data center. With the up-to-date map, T1 ESes serve as calibration points which map the reported incident onto absolute coordinates and notify nearby vehicles more efficiently.

3. **Data transmission:** The maintained map or other data, e.g., emergency or congestion information, can be transmitted between ESes via wired or wireless channels.

4. **Data dissemination:** Upon data updates, ESes disseminate data to vehicles in their coverage areas.

5. **Data aggregation:** T1 ESes aggregate data before sending it to T2 ESes for applications requiring larger amounts of data (naturally assuming aggregation is acceptable for the application).

6. **Data upload:** T2 ESes synchronize with cloud to update data and enable synchronization across a larger geographical area.

This communication model has several important benefits, namely:

1. **Neighbor notification** combines two methods of which the simple notification warns closest vehicles with the lowest delay, while the detailed report sends all information for ESes to generate AR data for CVV.

2. **Cache capacity** of an ES noticeably improves the performance of vehicular communication system. A common scenario is that vehicle detects an anomaly on road without nearby vehicles. The vehicle therefore cannot pass on the notification to other vehicles, but instead must upload it into the cloud. However, for a later vehicle to receive the notification in a timely manner, it needs to get the data all the way from cloud, or be in the coverage area of nearby RSUs when the data is still in transmission. ESes change this shortcoming by caching the data and broadcasting within predefined period, so that later vehicles receive the notification with lower latency.

3. **Hierarchical edge** enables efficient handling of workloads with different requirement by processing the data at the different tiers, depending on the application requirements..

### 5.5.4 Implementation

In terms of implementation, two key issues arise: the deployment and placement of ESes. Deployment refers to the internal implementation of an ES while placement refers to the physical placement of the ES.

**Deployment:** Our proposal to co-locate ESes with base stations and aggregation points are motivated by existing trends. The MEC standard developed by European Telecommunications Standards Institute (ETSI) proposes to deploy servers at the cellular base station to serve local mobile subscribers with fast response times [72]. Co-location with existing infrastructure also achieves cost-efficiency. For these reasons, T1 ESes co-located with base stations is a straightforward solution. Next, we need to take a look at cellular network deployment in near future to understand the rationale of T1 ES deployment. According to the 2017 survey of Small Cell Forum (SCF), by 2025, new non-residential small cell deployments will reach almost 8.5 million, which is 22 times higher than in 2015 [73]. On the other hand, 5G will also accelerate the deployment of small cells. 58% of the operators, according to the same survey of SCF, expect to focus primarily on small cells in the first 2-3 years of deploying 5G. However, the number

of macrocell seems to grow much slower. According to Nokia, traffic density of a very busy US city increased fourfold from 2004 to 2014, yet the average density of macrocell sites did not change [74]. We conjecture that small cell deployment will increase much faster than macrocell deployment in the near future. As a result, capacities of T1 ESes are facing increasing challenges and therefore we propose to locate the T2 ESes at a higher layer in the network to enable more efficient aggregation and backup.

**Placement:** To avoid unnecessary investment and complexity, the ESes location should be carefully determined. While T2 ESes are locate typically at aggregation points, which are relatively few in number, locations of T1 ESes have much more candidates, namely the macrocells. Cities like New York have macrocell deployments with 500 m inter-site distance or less [**?**]. Deploying one T1 ES per macrocell would be excessive in terms of investment, so to improve efficiency, we need to optimize the selection of locations in some manner. Our proposed algorithm includes two steps, namely average traffic clustering and edge capacity assignment. We opt for a hierarchical clustering algorithm since our edge layer already is constructed hierarchically. Edge capacity assignment is solved as a primary facility location problem, where we simply assign edge capacity to each cluster center (both Tier 1 and 2), proportional to its average traffic. The order of edge capacity is calculated through edge server capacity, traffic density, and resource consumption of the application (Section 5.7).

### 5.5.5 Use case solution Overview

Now we describe how to implement an ARHUD-based CVV. To solve the challenges described in Section 5.4, we need to implement the following components: 1. **Map maintenance:** Vehicles record the surrounding 3-D features with the coordinates of traversed streets and send to nearest T1 ESes. T1 ESes stitch together the collected segments to form 3-D neighborhood maps. 2. **Incident report:** Once a vehicle detects an incident, it sends the simple notification and detailed report to the nearest vehicle and T1 ES, respectively. 3. **Data process:** The ES extracts the data from the received report and localizes the incident in the map it maintains. The localization can use either the received coordinates or map the observed 3-D features within the map. 4. **Transmission and dissemination:** Meanwhile, the ES forwards the notification to nearby T1 ESes (directly or via T2 ES) and disseminates to vehicles within its coverage. The range of the dissemination area depends on the magnitude of the incident and the coverage area of the ES. 5. **Aggregation:** T2 ES aggregates data from T1 ESes to gather information of larger area. Use cases include, for example,

crowd-sourcing the neighborhood maps to build an urban 3-D map or traffic light control. 6. **Synchronization:** ESes synchronize with the cloud periodically or when triggered by specified incidents.

## 5.6 Network Issues

In this section, we discuss possible network protocols for V2E powered vehicle communication system. ARVE does not have any specific requirements on the networking technologies or protocols that are used. We can accommodate different technologies including cellular, Wi-Fi, D2D and DSRC so that they complement each other to fulfill different kinds of workloads and constitute an integrated networking system. Considering Figure 5.2 as an example, the device layer includes V2V and V2I communication where DSRC and D2D protocols coexist to provide better performance. Standalone D2D (Wi-Fi Direct) and DSRC could support V2V in scenarios even without network coverage. Another D2D protocol, LTE Direct, needs network assist and supports long distance connection. As shown in [65], Wi-Fi Direct has better performance than LTE and higher theoretical throughput than DSRC. However, WLAN chipsets are unlikely to fulfill ad-hoc communication at high speeds which makes them unreliable for vehicle network [75]. Here we propose to use a combination of D2D and DSRC to serve large volumes of data and fast data transmission, respectively. For instance, in our communication process, vehicle sends out the simple notification to closest vehicle by DSRC, while sending the detailed report to nearby ES by D2D. The rest of the system communicates via wired and LTE or 5G network. Today's cell phone connects to internet via cellular or Wi-Fi network, depending on local network coverage and subscription etc. Likewise, vehicular networks should also use multiple complementary protocols to function in different scenarios.

## 5.7 Preliminary Evaluation

In this section, we will present a primary ES placement solution for a CVV application based on ARVE and elaborate the system improvement over current vehicular network.

**Data Collection and Analysis:** To address the edge server placement problem, we study the base station and traffic distribution pattern in the center area of London as an example. The selected area has a size of 26km * 20km, and we collect the LTE base station location data[2] and traffic

[2] `https://unwiredlabs.com`

volume data[3] that fall into this area according to GPS coordinates.

First we cluster the traffic volume data according to their GPS coordinates, and divide the selected area into 20 small areas according to the clustering result. The traffic distribution and area partition results are shown in Figure 5.3, where each colored dot represents the location of the aggregated traffic, and the different sizes of the dots reflect the different traffic volumes in 12 hours during daytime. Next we want to see if base stations distribute differently from traffic, to understand if this would influence our co-located ES placement. There are 22041 LTE base stations located within this area, among which 1538 base stations have a coverage radius larger than 3000m, comparable to macrocell. We plot these 1538 base stations on the map, as shown in Figure 5.4. It can be easily observed that the base stations distribute evenly and reasonably match the amount of traffic in dense areas. As a result, using base stations as deployment points is not going to deviate the ES placement from the actual traffic patterns.

**Edge Server Placement:** H. Qiu et al. reported a typical Augmented Vehicle Reality system [65], where the AR related processing (e.g., point cloud manipulation) takes 1.337 sec on average. Considering this processing as the AR workload of the edge servers, one edge server is able to handle 2692 requests per hour, that is, serving around 32k vehicles during each daytime. The edge server placement is correlated with the traffic volume distribution, which is not uniform among the 20 small areas. The numbers of edge servers needed by each area are shown in Figure 5.5. In total 90 edge servers are needed in the selected center area of London and the largest clusters of ESes have a total of 8 ESes, while the bulk of them contain 3–4 ESes.

**Latency Comparison:** Edge servers bring the processing capability to the vicinity of vehicles. The latency of augmented vehicle reality consists of mainly two parts: the data transmission time and the server processing time. The data processing time taken by the edge server and the cloud server would not differ significantly, but the data transmission time is greatly influenced by the transmission distance. As reported in [65], the point cloud data size of the view generated by a 720P resolution stereo camera is 14.75 MB. Considering the edge server scenario, the uplink bandwidth between the vehicle and the LTE base station achieves on average 25 Mbps[4], so that the transmission of the point cloud finishes in 4.72 sec. On the other hand, the transmission between vehicles and cloud servers is

[3] `https://data.gov.uk/dataset/gb-road-traffic-counts`

[4] https://www.4g.co.uk/how-fast-is-4g/

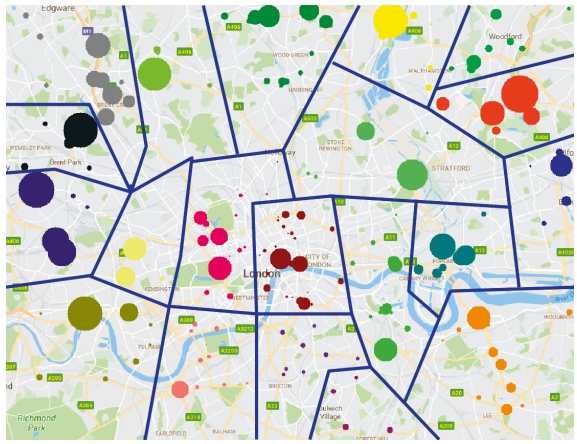



Figure 5.3: Traffic distribution in London.

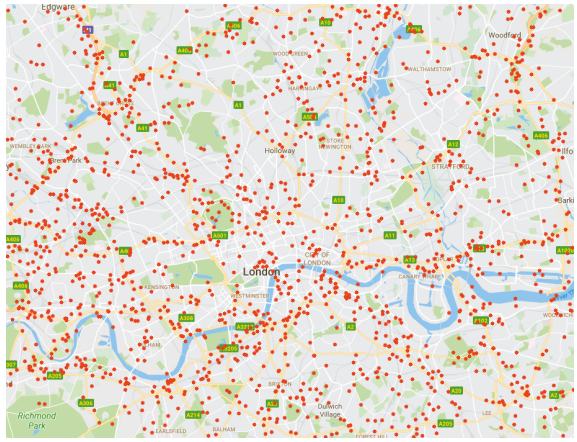



Figure 5.4: LTE base station (with coverage radius > 3000m) distribution in the selected area of London.

obviously slower, as the data needs to traverse through the Internet. Taking Google Cloud Platform as an example, the average uplink bandwidth is 4.4 Mbps[5], so that the transmission of the point cloud could take up to 26.82 sec. Our preliminary evaluation shows that ARVE would decrease transmission latency noticeably.

[5] `https://testmy.net/hoststats/google_cloud`

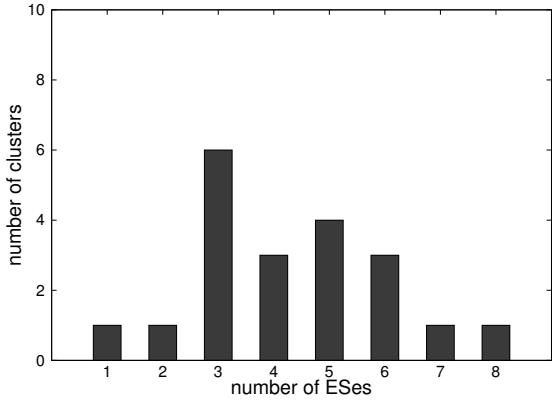



Figure 5.5: Number of edge servers needed by different areas.

## 5.8 Conclusion and Future Work

In this chapter, we have presented ARVE, an architectural framework for vehicle-to-edge applications. Our system could serve applications with different requirements in vehicular communication system. We choose CVV as the representative use case and proposed corresponding solution in details. With our preliminary evaluation using real data from London, we have shown that ARVE could improve vehicular network significantly with only reasonable requirements on the number of installed edge servers. In our future work, we will solve the specific challenges in CVV especially regarding ARHUD. The complex computational process involves several steps and may require orchestration of edge and vehicle resource, to improve utilization and computation efficiency while decreasing latency. We also plan to implement parts of the solution using real hardware and networking devices.

# Chapter 6

# Augmented Vehicular Vision: System Design

## 6.1 Overview

Vehicular communication applications, be it for driver-assisting augmented reality systems or fully driverless vehicles, require an efficient communication architecture for timely information delivery. Centralized, cloud-based infrastructures present latencies too high to satisfy the requirements of emergency information processing and transmission. In this chapter, we present EARVE, a novel Vehicle-to-Edge infrastructure, with computational units co-located with the base stations and aggregation points. Embedding computation at the edge of the network allows to reduce the overall latency compared to vehicle-to-cloud and significantly trim the complexity of vehicle-to-vehicle communication. We present the design of EARVE and its deployment on edge servers. We implement EARVE through a bandwidth-hungry, latency constrained real-life application. We show that EARVE reduces the latency by up to 20% and the bandwidth at the server by 98% compared to cloud solutions at city scale.

## 6.2 Introduction

Automated driving has recently gotten closer to becoming a reality. In 2018, California and Shanghai authorized the deployment of autonomous vehicles on public roads for testing purposes [55, 56]. In parallel to automated driving, manufacturers are constantly improving the assistance systems embedded inside vehicles. These systems nowadays heavily rely on environment sensing for signaling potential danger to the driver and

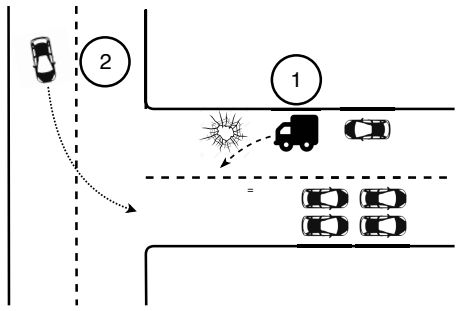



Figure 6.1: Common connected vehicles scenarios.

taking decisions if necessary. For instance, most manufacturers developed emergency braking systems for their top of the line vehicles. By combining information from the embedded radar and camera, the system can detect and prevent imminent collisions.

Although automated systems efficiently improve road safety, they are limited to the point of view of a single vehicle. However, some complex situations require assembling an aggregated point of view over several vehicles to avoid collisions. For instance, if the braking distance is too short to avoid a collision, the vehicle may choose another emergency maneuver such as steering into another lane. The system should request status from other vehicles in the area to assess the safety of the operation. Vehicular communication systems therefore play a key role in sharing information between vehicles and roadside infrastructure units (RSU). Current solutions focus on three types of communication: vehicle-to-vehicle (V2V), vehicle-to-cloud (V2C), and vehicle-to-roadside infrastructure (V2I) [57, 58]. Although these solutions fulfill basic demands, *efficiently sharing complex and large volumes of data among vehicles at scale* remains a challenge.

Figure 6.1 illustrates two common scenarios, with their relative latency and scale requirements:

1. *Real-time latency, street scale.* The leading truck encounters an unexpected pothole. The truck notifies the following cars to avoid a potential accident.

2. *Medium latency, city scale.* Congested traffic is out of sight for cars planning to take the road on the right. Vehicles in the congestion broadcast to all the network so that cars can compute another optimal itinerary.

| Edge | A | B | C |
|---|---|---|---|
| 19.9 ms | 24.9 ms | 52.4 ms | 58.8 ms |

Table 6.1: Average network round-trip latency over LTE to different targets.

Current V2V, V2C, or V2I architectures and solutions cannot handle these scenarios due to the diversity of the requirements. In V2C, the combined latency of transmission, processing, and distribution prevents emergency decisions to be propagated on time. On the other hand, although V2V significantly improves performance at close distance, forwarding information at city-scale is inefficient and costly. Finally, V2I provides better data distribution. However, sharing accurate emergency information entails non-trivial computation and coordination in a limited amount of time. Roadside infrastructures should therefore integrate computing features for fast and reliable emergency information propagation.

Edge computing facilitates latency-sensitive workloads by performing data processing in Edge Servers (ESes) located close to the user. The gain in latency provided by edge computing can be considerable. In Table 6.1, we measured the round trip latency for various servers through an LTE network. The first pingable IP is regarded as *Edge* server. We then consider the closest cloud servers provided by three leading companies in the market: A (local to the city - 5 km), B (nearby country – 1000 km) and C (A country further away – 2500 km). Unsurprisingly, the latency to the closest cloud server is half the round trip time to the furthest cloud server. The ES presents a 20% improvement compared to the nearest cloud server. These 5 ms may become vital in the case of road safety, especially considering that latency is cumulative in the case of information propagation. For a V2C application residing at either site B or C, the latency improvement is considerable.

In this chapter, we propose EARVE, the first *vehicle-to-edge (V2E)* framework to enhance vehicular communications. As an extension of our previous work [76], we propose the specific design of edge server and the interactions between server functionalities and the system (Section 6.4). We also implement a prototype and evaluate it with a preliminary experiment (Section 6.7). Our design integrates both cloud and edge computing capabilities for city-wide autonomous and semi-autonomous vehicle communications. First of all, we stress that EARVE network-agnostic in terms of the physical layer on top of which it is run. Currently, two main alternatives

exist for vehicular networks, Direct Short Range Communication (DSRC) and 5G [59]. While 5G seems to have its advantages [60], DSRC has already been adopted and deployed for tested solutions [61]. Since EARVE does not depend on any particular features of the underlying network, it can run on current (LTE) and future communication infrastructures seamlessly, while formulating recommendations for service provision and infrastructure deployment. As a concrete example, we apply EARVE to connected vehicle views by demonstrating the use cases, i.e., View Share, in Section 6.5.

This chapter makes three key contributions.

- We present the design of EARVE, which equips edge RSUs with computation and cache capacity. We describe the implementation details and communication flows within the edge servers and between them.
- We describe a concrete application of EARVE to connect vehicle views using Augmented Reality Head-up Display (ARHUD). This use case displays the advantages of EARVE while scaling the problem of vehicle vision from a network perspective.
- We present an evaluation to show that EARVE meets the performance targets we have defined and that it offers noticeable performance improvements with reasonable expenditure in infrastructure.

The rest of this chapter is structured as follows. In Section 6.3 we present the overall system design. Section 6.4 provides the detailed description of the implementation of an edge server in EARVE. Section 6.5 presents the two concrete use cases we consider in the chapter. We give practical details of our implementation in Section 6.6 and present our evaluation in Section 6.7. We cover related work in Section 6.8. Finally, Section 6.9 concludes the chapter.

## 6.3 System Design

In this section, we describe EARVE's design. First, we describe our proposed architecture for EARVE. We then discuss the major communication processes and propose a deployment scheme. Last, we discuss how EARVE can improve privacy and security in vehicular networks and AR in general.

### 6.3.1 System Architecture

EARVE is defined around three key layers: **device, access network and core network** as shown Figure 6.2. The **device layer** includes the vehicles, roadside buildings, infrastructures, phones of pedestrians and any

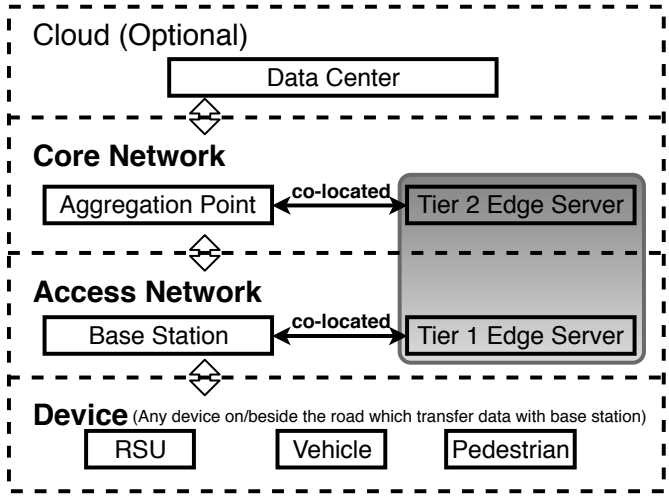



Figure 6.2: EARVE System Model.

other devices involved in the vehicular network. In the rest of this chapter, we will call "client" any object in the device layer that transfers data to the ES. The **access network and core network layers** host the ESes at the core of our architecture. We distribute these ESes hierarchically in two tiers. The first tier (T1 ES) consists of ESes co-located with the base stations at access network level[1], while second tier ESes (T2 ES) are co-located with aggregation points in the core network. Finally, we define an optional cloud layer to provide on-demand backup capacity.

The ESes communicate with the vehicles and RSUs via the radio access network, and backs up data with the remote cloud if necessary (e.g., map updates). T1 ESes operate within an area defined by the range of their corresponding macrocell and eventual small cells (see Section 6.3.3). As they are closer to vehicles, T1 ESes serve latency-sensitive applications such as emergency notifications. T2 ESes collect data from multiple areas (multiple T1 ESes) to provide a larger scale of service and data backup, e.g., to improve traffic flow by sending cruise control messages or controlling traffic light.

### 6.3.2 Communication Process

The communication process of EARVE follows 6 basic steps: *neighbor notification, data processing, transmission, dissemination, aggregation, and*

[1]Base station in this chapter refers to the entity at the edge of the fixed network, e.g., BTS, eNB and gNB.

*uploading.* In order to showcase this process, we consider an emergency notification application.

**Neighbor notification:** Emergency notifications need to be dispatched to the nearest clients with the lowest possible latency. The device detecting the emergency sends a detailed report to the nearest T1 ES. The report compiles the sensor data at the time of the incident with minimal compression to avoid losses and reduce processing times. Sensor data includes image frames, coordinates, velocity and motion direction of the device to help the ES calculate the coordinates of the incident.

**Data processing:** Once a T1 ES receives a report, it processes the data and caches the extracted information for passing on to later vehicles. As discussed in [65], sharing the views of incidents among vehicles is non-trivial. ESes maintain an up-to-date local map in real time, by collecting data from passing vehicles. This local map is then regularly synchronized with a cloud data center. T1 ESes serve as calibration points where the reported incident is mapped onto absolute coordinates before notification. ESes rely on machine learning based image analysis modules for extracting useful information from the reports. This data is then compared to a predefined rule set to determine the resulting actions. In the case of emergency notifications, ES compares the type and severity of the emergency with the rules and triggers the corresponding actions upon match.

**Data transmission:** Depending on the severity of the emergency, the T1 ES sends the notification to nearby T1 ESes on a scale defined by the matching rules. For instance, T1 ESes located within the same neighborhood are notified of events that may cause serious congestion.

**Data dissemination:** The top priority of each ES is to notify nearby vehicles and pedestrians. An ES needs to notify different groups of clients according to the triggered rules. For example, an ES notifies only the nearby vehicles of a "traffic congestion" event, while it notifies the nearest vehicles and pedestrians of a "severe accident" event.

**Data aggregation:** T1 ESes send data to T2 ESes for applications requiring larger amounts of data (only if aggregation is acceptable to the application). For instance, burst water pipes may cause severe flooding in multiple blocks. To get the whole picture, T2 ES needs to aggregate data sent from all involved T1 ESes within the damaged area.

**Data upload:** T1 and T2 ESes synchronize with the cloud to keep city-scale data up-to-date. T1 ESes also forward the emergency data to cloud for backup. This data may be used later for deploying new city-scale road security policies.

### 6.3.3 Deployment

Due to the space limitation, we will skip the details of deployment. Please refer to our previous work for more details [76].

### 6.3.4 Privacy and Security

V2V communication exposes each vehicle's private information to others. The mobility pattern of a driver is easily discovered which raises security and privacy concerns. Anonymization can hide the true identity of the sender, however, it raises issues in trust in the information shared in V2V networks. As a result, it is not easy to protect user privacy while providing trustful information distribution. Our system provides a preliminary solution for those problems with the help of ESes. As the information collection and distribution point, an ES collects data directly from users and distributes it after removing the private information of the original sender. As such, users share valuable information while hiding their identity from the larger public. Besides, our edge service follows a subscription mechanism, therefore only the users who trust and willing to use the edge service will share their information.

## 6.4 Edge Service

In this section, we describe the architecture of the services deployed at the edge. We first characterize the major data flows between server microservices. Then, we discuss the multithreading data process of ES and client device.

To achieve low enough latency, ESes need an efficient data process flow with only the key microservices. On the other hand, the architecture needs to remain flexible and allow adding more microservices in the future. We design our ESes as shown in Figure 6.3. This architecture spreads among four major planes defined as follows:

The **Device Service** layer is the interface through which the ES communicates with different devices. Any client in need of edge service communicates with ES through this layer. At this layer, the major microservices include *Device Discovery*, *Registration* and *Communication*.

- *Device Discovery* allows the automatic discovery of devices entering the coverage area of the ES. We propose a subscription model so that the ES only communicates with the clients willing to use the edge service. The edge service provider reserves a specific IP address for service data transmission. Any device demanding edge services sends

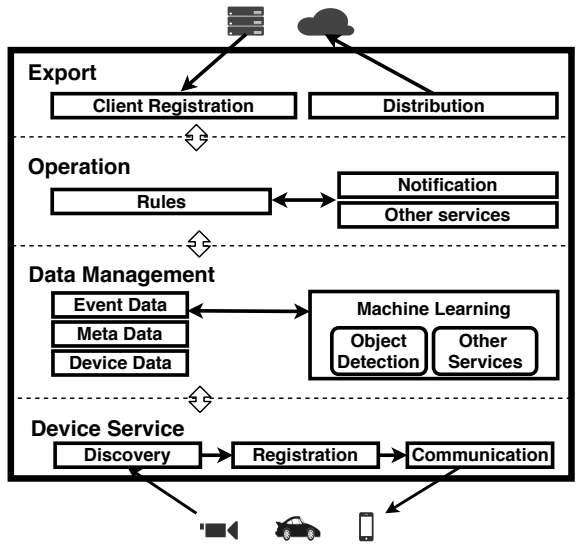



Figure 6.3: Edge Server Design.

its basic information (e.g., ID, profile and coordinates) to the specific IP address periodically. The information is included in a message similar to the "HELLO" packet in OSPF for dynamic discovery of neighbors. The T1 ES within the base station identifies the information by filtering the destination IP address and registers the device. Users can adapt the frequency of messages and turn it on or off freely. We choose to use this model to give the user more flexibility.

- *Device Registration* generates a *unique ID for the discovered device and asks for its profile and value description.* The device profile is a descriptive file including all basic information of the device. This profile is composed of key-value pairs that describe the parameters potentially sent by the vehicle. For instance, a vehicle registers with the following description: {*V: velocity, mile/h; D: motion direction*}. Later the same vehicle going southbound at 30 miles/h will send {*V: 30; D: south*}.

- *Device Communication* transfers data with the device. The formats of data include plain text and image frames. The transport protocol is flexible and depends on specific use cases. In this work, we use UDP for low latency image transmission. However, for other applications, it may be needed to develop additional mechanisms to ensure reliable and ordered transmission while keeping the low latency requirements.

| Timestamp | Coordinates | Type | Data | Device ID |
|---|---|---|---|---|

Table 6.2: Event entry.

The **Data Management** layer analyzes and manages the data received via the Device Service layer. The major microservices include *Device Data*, *Meta Data*, *Event Data* and *Machine Learning*:

- *Device Data* stores the basic device data such as UID and profile for the ES to uniquely identify the device.
- *Meta Data* stores the values description and other metadata.
- *Event Data* stores the up-to-date events sent by devices. The database stores events as unique sorted entries (see Table 6.2). The entries are organized in multiple levels and sorted from left field to right field (at the exception of "Device ID"). We base this ordering rationale on the scenario where multiple vehicles, pedestrians, and RSUs witness the same event or accident at the same time. To uniquely identify the event and ignore the duplicated reports, the "Timestamp" and "Coordinates" of the event have the highest priority for sorting. The "Type" field defines different types of events from normal map update to a severe accident report. Each event has a predefined TTL based on its type and gets removed from the database after expiration.

  The "Data" field contains all the key-value pairs from the device report. The "Device ID" is the unique ID of the sender and not included in the order of sorting, to avoid duplicated reports of the same event from different devices.
- *Machine Learning* analyzes the image frames sent by devices, extracts the key information and forwards it to the Event Data microservice. In this work, we integrate the machine learning algorithm of object detection into ES, which runs on GPU.

The **Operation** layer generates rules and applies them to the event data. The ES triggers a rule's actions if an event data matches the rule's condition(s). The major microservices include *Rules*, *Notification* and *others*:

- The *Rules* microservice generates new rules on-demand. Each rule has one or multiple conditions and actions. The ES matches each event data entry with the rule set and triggers its actions in case of a match.

Upon detection, the ES triggers the transmission of notification and the export of data to neighbor clients.

- The *Notification* microservice sends out alerts/notifications to specific clients (groups) defined in the rule(s). For instance, if the event data contains an anomaly, the "Notification" service is triggered and the ES sends out notifications to all devices defined by the rule, e.g., all vehicles discovered within 5 minutes.

The **Export** layer is responsible for exporting data to other ESes (T1 and T2) or the cloud if necessary. The major microservices include *Client Registration* and *Distribution*:

- *Client Registration* provides the interface for northbound clients to register to the system. In our system, northbound clients include nearby T1 ESes, T2 ESes, cloud data center and emergency center etc.

- *Distribution* sends to subscribers the event data based on the rules or historical statistics for backup.

Figure 6.4 depicts the major data process flow in the ES, which includes the following steps:

1. An ES discovers a device entering its coverage area.

2. The server registers the device and stores its basic information. The necessary information includes UID of device (generated by the ES), device profile (brief description of the device) and value description (key-value pairs to explain the meaning of value possibly sent by the device). The UID and profile of the device is stored in the Device Data microservice while value description is stored in Meta Data.

3. The ES communicates with the devices via the Communication microservice. It identifies each device by matching the device information with the corresponding Device Data. The ES allows a device to send plain text and image frames to report on different kinds of events.

4. Upon receiving plain text, the ES extracts the event data directly from the text and stores it. Upon receiving image frame, the ES analyzes it through machine learning algorithms and extracts the event data. In this chapter, we only focus on one application of image processing: object detection.

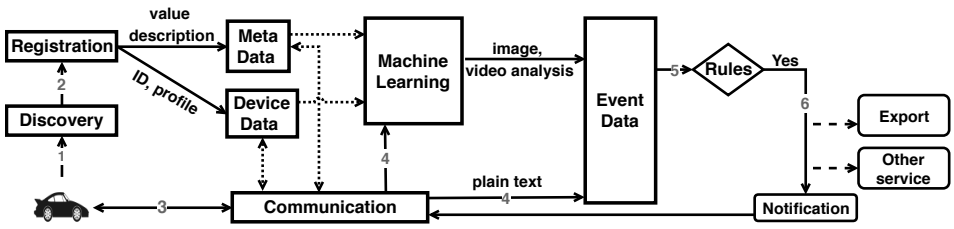



Figure 6.4: Data flow of Edge Server.

5. After extracting the event data, the ES matches it with the rules and eventually triggers the corresponding actions.

6. The actions include notifications to vehicles, pedestrians or emergency centers, propagation to nearby ESes and cloud backup etc.

## 6.5 Augmented Reality Applications for vehicular network

In this section, we describe the details of an AR application realized by V2E to showcase EARVE. We select an in-vehicle IoT device (e.g., HUD, smart rearview mirror) as our client device, and assume there is a T1 ES nearby the vehicle client and needs to provide object detection services.

**View Share:** To detect objects and achieve augmentation of the views, the client periodically uploads the sensor data and captured image frames to the ES, including the current street view image from the camera, GPS coordinates of the vehicle from the GPS receiver, the orientation of the vehicle from the IMU, and the timestamp. Together with the IoT device's IP address, the {*camera image, GPS, orientation, timestamp, IP*} tuple is the major input of our system. Upon receiving the sensor data, the ES first executes object detection on the camera image with the object detector. With state-of-the-art deep learning frameworks and GPU hardware acceleration, the object detector is able to detect objects in real time. For each detected object, a rectangular boundary is also given by the detector.

With the object detection results, the ES calculates the GPS coordinates of the detected objects. The first step is calculating the relative positions of the objects from the vehicle. For each object, we choose the middle point of its bottom boundary as its position in the camera image. We need to transfer this position in pixels into position in meters, which is the object's position in the real world. To achieve this, we change the

perspective of the camera image so that the new pixels correspond to the bird's-eye view from the top of the street. With the object's 2D position in the bird's-eye view image, we can calculate the object's relative position (both distance and direction) from the vehicle in meters after some calibration. With the vehicle's GPS information, the ES calculates the GPS for each detected object. Then all information is extracted as an event entry embedded into Event Data database. ES processes the data, applies the corresponding rules, and sends back {*timestamp, object name, object GPS*} tuples to the clients specified by the rule's actions. When the client receives the message from the ES, it displays the detected objects in a manner of AR. With the object's GPS, the vehicle's GPS and orientation, the client is able to calculate the relative position (both distance and direction) of the object from itself. This relative position is transformed into 2D on the screen after perspective and unit transformation. With this design, drivers are able to see the objects that are hidden by front vehicles in real-time in an AR manner.

## 6.6 Implementation

In this section, we describe the implementation details of our system. We follow the use case described in the previous section and develop a simple object recognition system for vehicles. This system detects pedestrians and cars on pictures sent from the client device's embedded camera and returns the results in real-time. Our client is implemented on the Android platform, simulating the hardware and software environment of the vehicular equipment for augmentation. The GPS sensor reports the GPS coordinates of the vehicle, and the monocular camera captures the front-facing view from the vehicle. OpenGL is utilized for rendering the augmented information on top of the camera view. The communication between client and server is based on sockets with all the information packed in our own formats. The plain text and image frames sent by the client, as well as the message sent by the server, are transmitted over UDP socket for low-latency transmission.

Our server is deployed on a Linux platform. For object detection, we utilize the GPU implementation of YOLO version 3, which is the state-of-the-art object detector. We use OpenCV for general image processing like perspective transformation (from one vehicle's to another one's). For *ES implementation*, we build our prototype on top of EdgeX Foundry project, a vendor-neutral open source framework for IoT edge computing [77]. We use EdgeX as the skeleton framework with proper adaption and add more

microservices to build the ES for EARVE. For instance, to have the best knowledge of up-to-date events, we change the database maintenance mechanism of *Event Data* to multi-level uniquely sorting. We add the device discovery microservice and tune the communication module by adding UDP socket method. Moreover, we integrate machine learning modules into the ES architecture to improve its data analysis ability.

## 6.7 Evaluation

We built a proof-of-concept system and now present our evaluation. To emulate an in-vehicle IoT device, we installed our client on a Xiaomi Mi5 smartphone, with a 2.15GHz quad-core Snapdragon 820 CPU and 4GB of memory. Our ES is deployed on a local Linux PC, with a six-core i7-5820K CPU, 64GB of memory, and an Nvidia GTX 1080Ti GPU. The hardware specification of our ES is similar to a medium-priced edge server in 2018 [78]. To compare the benefits of EARVE to cloud computing, we create a virtual machine instance on the Google Cloud platform, with 6 vCPUs, 16GB of memory, and an Nvidia K80 GPU. The phone is connected to the Internet through a WiFi access point on the ES. As such, our ES deployment works also as a "base station" from the perspective of the phone client, which follows our deployment proposal, namely co-locating ES with the base station.

The RTT from the phone to the ES is 6.84ms, and the RTT from our ES to the Google cloud virtual machine is 28.76ms. More than 90% vehicles driving at 100 km/h have only 7.6 milliseconds RTT in LTE network [79]. Besides, vehicles normally drive much slower in urban areas, therefore our evaluation setup represents a realistic LTE vehicular network approximation. The measurements shown in Table 6.1 are also in line with our setup's ES-to-cloud RTT. Although being an indoor evaluation, our setup is close to vehicle networks' reality and validated for testing performance in real scenarios.

### 6.7.1 ES placement

To address the ES placement problem, we consider the base station and traffic distribution patterns in central London. The selected area has a size of 3.91km * 5.75km. For this area, we use public LTE base station location data[2] and traffic volume data[3]. We cluster the traffic volume

[2] `https://unwiredlabs.com`

[3] `https://data.gov.uk/dataset/gb-road-traffic-counts`

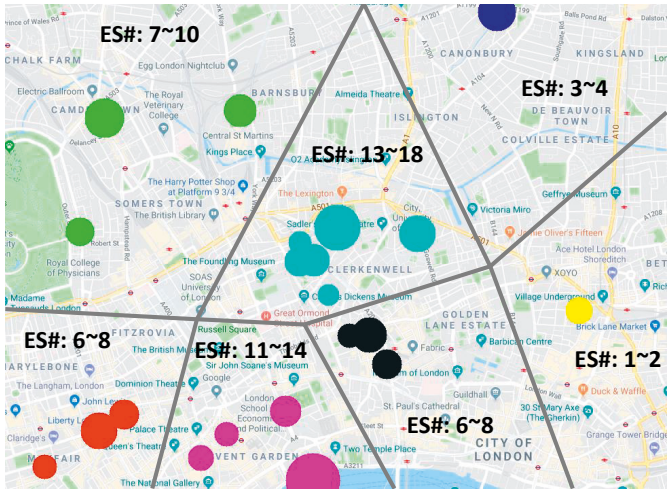



Figure 6.5: ES placement based on traffic heatmap.

data according to its GPS coordinates and divide the selected area into 7 small areas according to the clustering result. The traffic distribution and area partition results are shown in Figure 6.5. Each colored dot represents the location of the aggregated traffic, with size proportional to the traffic volume in 12 hours during daytime. In each area, we display the number of deployed ESes, and average and the peak traffic volume. We evaluate both cases to have a better understanding of ES placement's influence on system performance (see Section 6.7.3).

We then analyze the relationship between the base station distribution and the traffic density, as it influences our co-located ES placement. There are 3455 LTE base stations located within this area, among which 81 base stations cover more than 3000m, comparable to a macrocell. We plot these "macrocells" in red and the others in blue, as shown in Figure 6.6. The base stations are distributed evenly and reasonably match the amount of traffic in dense areas. As a result, using base stations as deployment points is not going to deviate the ES placement from the actual traffic patterns. The maximum number of ESes needed to meet the user demand during peak traffic is 64. Thus, we deploy ESes within the macrocells which are closer to the location of the aggregated traffic.

### 6.7.2 AR Applications

We then carried out experiments into the data processes in our representative AR application: View Share. We ran the application using ES and

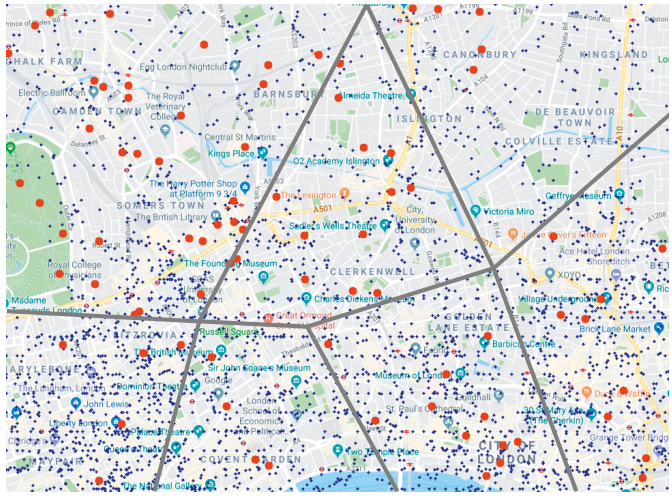



Figure 6.6: LTE base station (in red, the ones with coverage > 3000m) distribution in the selected area of London.

cloud for 1s (View Sharing) over 100 runs and measured the step by step latencies. Each run includes the entire integrated workflow of the application (see Section 6.5). The result is shown in Figure 6.7.

The total latency for View Share can be divided in the following segments: *Client Data Collection*, *Uplink Latency*, *Object Detection*, *Policy Control*, *Downlink Latency* and *Client Rendering*. As Figure 6.7 shows, View Share with ES is 32.9ms faster than with the cloud. Due to the hardware difference, algorithms run slightly faster on ES than on cloud. Nevertheless, the sum of uplink and downlink latencies decreases by 21.7ms, which contributes to most of the improvement. This represents a 20% improvement over cloud computing, with an overall latency under 100ms. This proves our core assumption that edge computing can significantly improve latency-sensitive workloads by performing data processing closer to the user. It also addresses part of our second challenge (Section 6.2): edge computing improves AR application in vehicle networks, on the granularity of single workflows.

### 6.7.3 Scalability

To evaluate the scalability of our system, we test the bandwidth requirements and average latency for different vehicle traffic densities. We reuse the setup described in Section 6.7.1. We select traffic data from 7am to 6pm (12 hours), during which the number of passing vehicles increases

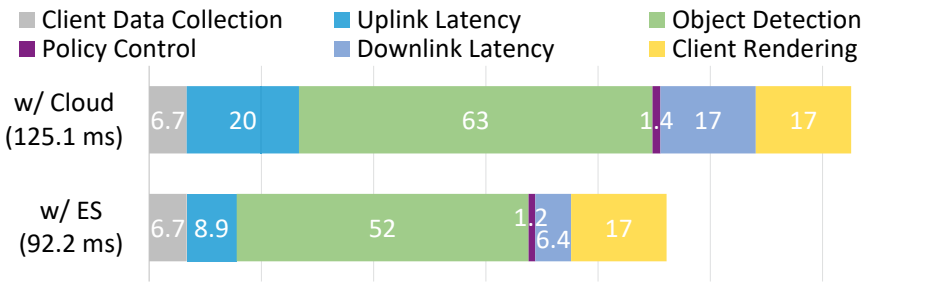



Figure 6.7: View Share Latency Decomposition.

| | Device | Edge Server | Cloud Server |
|---|---|---|---|
| Min | 0.0581 MB/s | 2.89 MB/s | 184.9 MB/s |
| Max | 0.0581 MB/s | 5.07 MB/s | 324.5 MB/s |

Table 6.3: Minimum and Maximum bandwidth requirements at device, edge and cloud level for EARVE during a day.

from 8262 to 14494 and has a peak at 6pm. Each ES has a coverage area of 3km, similar to its co-located macrocell. The average vehicle speed is 36km/h (10m/s). Each vehicle spends on average 5min within the coverage area of a given ES. The public dataset we used only contains vehicle count per hour. Here we build a vehicle distribution model based on Normal distribution. In our experiment, we use 5 minutes (300 s) as the cycle of this normal distribution, with a mean of 150 s and variance of 150 s, and the same pattern repeats 12 times for an hour. For every 5 minutes, the number of vehicles within the coverage of ES changes with time, as the first-appear time and the speed of these vehicles vary. Each vehicle sends captured images continuously to the nearest ES or cloud at 30 fps. Out of the 30 frames per second, the first frame is 960 x 540 pixels (51.8 KB) which is used for object detection.

In Table 6.3, we display the minimum and maximum bandwidth requirements of our demo application over the day.During peak hours, a cloud server running such operations would require more than 2.5 Gb/s for a single software. On the contrary, when using edge server, the load at the co-located base station is around 40 Mb/s with 0.465 Mb/s per device, which is achievable on LTE networks, and considerably relaxes the overall load at the bottleneck (up to 98%).

Next, we test the delay of the AR application, under different ES deployments and traffic densities along the 12 hours. As shown in Figure 6.8, we tested 6 ES deployments with different numbers of ESes. The different

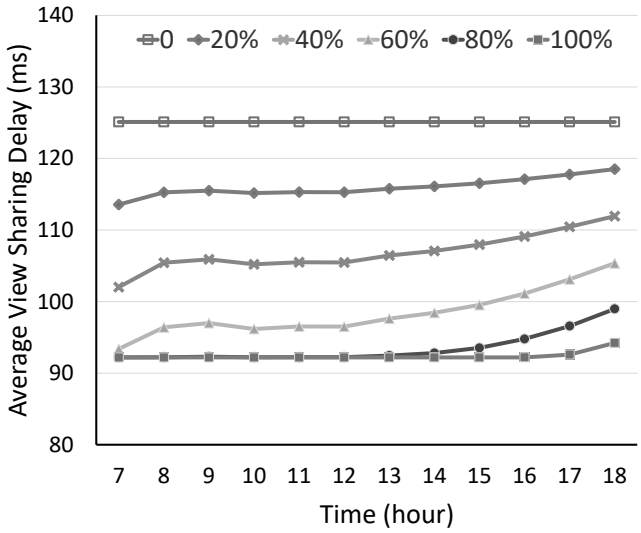



Figure 6.8: Average latency in different time periods for various load distribution between ES and cloud (100% is Edge only).

deployments are defined by the ratio of "summation of edge capacity" to "overall demand". The definition of "edge capacity" depends on specific service provisions. For instance, the period (T) of object detection is $1s$. Then a single ES has capacity calculated as follows,

$$C_{object} = T/delay = 1000(ms)/52(ms) = 19$$

which means an ES can process 19 tasks of object detection (on GPU) *within the corresponding period*, parallelized. Based on this definition, we defined following ES deployments. "0" is the pure cloud solution without any ES deployment, and we extend the cloud server resource to make it capable of fulfilling all the requests (provides the optimal performance for a cloud solution, to be compared with ES solutions). "100%" is the pure edge solution when the overall capacity of deployed ESes can fulfill the peak demand *on average*, e.g., there are 14494 passing vehicles during the hour around 6pm in the area, the average number of object detection tasks received by each ES during this hour is 1200 per second, which is calculated as follows:

$$N_{r/ES} = \Delta N_v * T * f = 14494/3600 * 300 * 1$$

where $\Delta N_v$ is the incremental number of discovered vehicles per second, $T$ is the time a vehicle crosses the coverage of an ES, $f$ is the request frequency.

In this particular hour, it requires 64 ESes (1200/19) to fulfill the average demand (as shown in Figure 6.5). With this setup, pure edge solution can fulfill *most* requests during the peak hour, except for those peak time points when significantly more than the average number of vehicles sending requests simultaneously. The other deployments are mixtures of edge and cloud solutions, where ESes can fulfill specific percentages of the requests and forward the rest to the cloud, e.g., "80%" represents when deployed ESes can fulfill 80% of the requests and the rest 20% are sent to the cloud. The result in Figure 6.8 shows that deployments with more ESes have lower delays. Comparing with the pure cloud solution, the pure edge solution decreases delay by 32ms for View Sharing in most periods, while the others also decrease delay at different levels. "80%" deployment gets similar delays with the pure edge solution.

In summary:

1. Our ES placement proposal follows the practical traffic and base station distribution.

2. EARVE improve the AR applications in vehicle networks by decreasing the transmission latency.

3. EARVE is scalable and performs well in different traffic densities. It can also be combined with cloud solutions to optimize the costs.

## 6.8 Related Work

Emerging technologies enable various functions for autonomous vehicles but also bring new challenges.

**Network protocols:** Direct Short Range Communication (DSRC), Device to Device (D2D) and 5G, improve data transmission [59–61]. However, the large volumes of data will challenge current computation resource deployments and risk making them bottlenecks [80]. In this chapter, we focus on V2E communication and select LTE as the network protocol. The rationale is straightforward, it accords with our scheme that ESes are co-located with base stations. Moreover, the major network workloads of our system and AR application, are image transmission and notification broadcast (or multicast depending on the rules). LTE outperforms DSRC in both workloads because of its longer coverage range and throughput performance, as shown in work [10].

**Edge Computing:** Edge computing to bring computation close to the user has attracted attention, such as [62] which explores integration

of 5G, SDN, MEC and vehicular network. Uncoordinated strategies for edge service placement have been investigated in [63] and the results have shown that they work well for this problem. Meanwhile, the fundamental issues, i.e., architecture design, communication process, network protocols and implementation concerns are yet to be explored.

**Applications:** Efforts on developing vehicular applications have achieved some results [65, 66], but without improvement from system and networking point of view, those applications face difficulties to scale in realistic situations.

## 6.9 Conclusion

In this chapter, we present EARVE, an architectural framework for vehicle-to-edge applications. Our system exploits the low latency of Edge servers to provide real-time emergency detection and notification. Thanks to its layered architecture, EARVE provides servers at street, neighborhood, and city that allow for a variety of usages at different scales in time and space. EARVE also presents the advantage to be mostly agnostic to the network and the hardware of vehicles and offloads most of the computations to ESes. To validate the concepts behind EARVE, we build a prototype application that we evaluate through both simulations and real-life conditions. Using real traffic data from London, we show that EARVE improves vehicular network significantly with reasonable requirements in terms of number of installed edge servers. Our evaluation results show that, compared to cloud solutions, EARVE decreases the latency of AR applications in vehicle newtorks, e.g., 26.3% for View Sharing. We also investigate the scalability of EARVE and show that it decreases latency in realistic scenarios for different traffic densities. We test mixed edge and cloud solutions and find out that more ES deployments bring larger improvements. In a word, EARVE is an efficient V2E solution which improves the performance by decreasing user latency and reducing network traffic.

# Chapter 7

# Mobile AR Multiple server offloading

## 7.1 Overview

The evolution of mobile devices enabled the development of sophisticated and resource-demanding mobile applications. As a representative example, Mobile Augmented Reality (MAR) applications employ computationally demanding vision algorithms on resource limited devices. Computation offloading on cloud or edge servers overcomes this resource limitation at the cost of additional network delays. The use of multiple paths at the same time has been proposed to overcome network limitations but it is not easily adaptable to the edge computing paradigm due to the server proximity. In this article, we propose an extension to current mobile edge offloading models using multiple paths for computation offloading. We solve the problem of offloading computations over several servers and links and derive the model to device-to-device, edge and cloud offloading in parallel. We then introduce a new task allocation algorithm exploiting this model for MAR offloading. Finally, we evaluate multipath offloading compared to single path models. Our system achieves 100% in-time completion in scenarios where single path struggles to keep the excess latency to acceptable levels. We also measure the impact of the variation of WiFi parameters on task completion, as most of the traffic will be directed through this network. We finally demonstrate the robust fallback provided by our system in case of network instability: 96% of the tasks are completed in time with only 70% WiFi availability.

| D2D | Edge | Edge | Alibaba | Alibaba | Google | Google |
|---|---|---|---|---|---|---|
| WiFi D | WiFi | LTE | WiFi | LTE | WiFi | LTE |
| 3.5 ms | 3.7 ms | 19.9 ms | 5.5 ms | 24.9 ms | 42.2 ms | 52.4 ms |

Table 7.1: Average network round-trip time measured for different offloading mechanisms.

## 7.2 Introduction

**Motivation:** Over the range of existing multimedia applications, Mobile Augmented Reality (MAR) may be the most demanding in computational resources. A typical MAR application processes large amounts of data (typically video flows) to display a virtual layer on top of the physical world. These operations are usually performed on mobile devices such as smartphones or smartglasses that can only execute basic operations. More advanced functions, such as environment analysis have to be performed on an external server. In the last decade, the increase in the performance and ubiquity of both servers and networks, enabled executing larger portions of code on remote devices in the cloud and at the edge of the network. In parallel, other technologies such as Device-to-Device (D2D) communication lead to a further decrease in latency and battery usage caused by network communication. However, the introduced network delays are non-negligible for latency-constrained applications.

**Network latency impact:** Table 7.1 presents the round-trip times (RTT) measured between a smartphone (LG Nexus 5X) and several potential offloading devices: another smartphone connected using WiFi Direct (1m distance), an Alibaba Cloud virtual machine through WiFi (via *eduroam*[1]) and LTE, a Google Cloud virtual machine through WiFi and LTE, as well as the first reachable server to emulate an Edge server. We average our measurements over 100 ICMP packets. The latency increases dramatically with the distance between the client and the server. D2D shows RTTs ias low as 3.5 ms. The WiFi access point several meters away adds 0.2 ms, and the Alibaba cloud server 2 ms. Although Google Cloud does not have servers in our city, we wish to show the effect of a more distant cloud provider – about 1,000km away. Latency is multiplied by 8 compared to the local Alibaba server. LTE also adds noticeable latency relatively to WiFi: 16 ms for an Edge server and 10 to 19 ms for a Cloud server.

[1] https://www.eduroam.org/

MAR applications are both CPU hungry and latency sensitive (some tasks have to be executed in less than 20 ms [81]), and may run on constrained devices such as smartglasses. In-time task completion is therefore intricate to achieve, both on the client due to the low computing power, and on the server due to the added network delays. As we can see Table 7.1, for a task with deadline 20 ms, offloading it to a nearby smartphone leaves less than 16.5 ms for the actual task processing. On the other hand, offloading to a local cloud server provides less time for processing (14.5 ms), but offers computational power several orders of magnitude higher. As the latency added by the network may account for more than half the task completion deadlines, maximizing in-time task completion rates involves minimizing the transmission related delays.

In consideration of the above constraints, achieving timely processing of a 30 frames per second video flow for a MAR application represents a challenge that requires the use of the multiple links and servers available in the system. To provide such performance, MAR applications should not only dynamically offload their computations in parallel over the set of available devices, but also exploit the multiple available links to minimize link and transmission delays.

**Contributions** In this chapter, we consider a heterogeneous environment composed of servers ranging from a companion smartphone located one hop away to an array of cloud servers in a different Autonomous System. After adapting a MAR application for multipath offloading, we model the effect of the network latency and bandwidth on task response time. Using this model, we devise a scheduling algorithm for multipath task allocation. We evaluate this scheduling algorithm through both a simple demonstration application and extensive simulations. After comparing our solution to single path task allocation, we analyze the impact of both the access link as well as the computing power of servers on the task distribution and the in time task completion. Finally, we test the robustness of our model to the instabilities inherent to wireless links.

Our contributions can be summarized as follows:

1. *Multipath offloading model.* Model network and servers impact on the task latency in a multipath scenario.

2. *Maximize task completion.* A scheduling algorithm to allocate tasks in a wireless multipath environment.

3. *Real-life Implementation.* Perform a computation-heavy computer vision task on two distinct servers in parallel.

4. *Characterize network impact.* Evaluate the sensitivity of MAR applications to the network characteristics of access links. We show that the system is very sensitive to bandwidth variations, but not so much to latency.

5. *Fallback evaluation.* Highlight the spillways provided by such a heterogeneous system to overcome link and device instability or failure with minimal impact on the application's timings.

## 7.3 Related Works

Computation offloading dates back from the earliest ages of computer science, and was one of the main motivation for computer networks. In a memo considered as the first documented evocation of computer networks (1963), J.C.R Licklider [82] justifies the need for device interconnection to enable access to distant computing resources. In more recent years, the explosion of the mobile device market shed new light on these problems, introducing new needs and constraints. Many cyber-foraging solutions for mobile applications were developed, whether in the cloud [83, 84], the edge of the network [85], or exploiting D2D communication [86].

Offloading frameworks have been proposed to enhance the capabilities of hardware-limited mobile devices. These frameworks focus on the data partitioning problem as well as its implementation in current mobile devices. MAUI [87] focuses on the energy consumption of mobile devices to perform offloading. The developer of the application provides the data partitioning through a simple annotation system. CloneCloud [88] modifies the application layer virtual machine to automatically offload tasks to a remote server with no intervention on the application. ThinkAir [89] focuses exploiting the elasticity and scalability of the cloud. This framework distributes offloaded functions in parallel among multiple virtual machine images in the cloud. Cuckoo [90] is another generic framework aiming at offloading computation with minimal intervention from the application developer. This framework was implemented and evaluated as part of an AR application.

Several other studies were directly focused at AR-specific offloading. Back in 2003, Wagner et al [91] proposed to offload AR tasks in the cloud. Shi et al [92] provide guidelines for MAR applications on wearable devices. Finally, Overlay [93] exploits cloud offloading for AR applications. However, these works focus on pure cloud or edge computing, with eventual distribution over several servers positioned at the same level in the network. Moreover, these works neglect LTE links due to their high latency

and variance. One of the first goals of our study is to analyze the impact of offloading computations to servers located at different levels of the network, and evaluate the effect of the access link on the resource allocation.

In parallel to these generic and AR-specific cloud offloading frameworks, new applications were developed, exploiting either D2D or edge computing. D2D communication is defined as the direct communication between two mobile devices [94, 95]. D2D communication has been used for offloading over Bluetooth [96], WiFi Direct [97], or even NFC [98]. Mobile edge computing is considered as an extension of the current cloud model, in which the servers are brought as close as possible to the access point to reduce network-induced latency and avoid congestion in core network. This new paradigm attracted a lot of attention, not only from academia [99–101], but also the industry [102, 103]. We integrate both paradigms in our model in association with cloud computing, as their distinct characteristics can prove essential for enhancing the experience of offloading applications.

More recently, some studies started to focus on the networking aspects of computation offloading, whether from an energy perspective [104], to reduce data transmission [105], or optimize mobility [106]. The pure networking challenges of AR have been evoked through several articles. [5] proposes to combine AR offloading and Information Centric Networks to optimize data transmission. [107] focuses on the application layer networking constraints, while [80] insists on the possible transport layer optimizations. In this chapter, we take into account these network-oriented studies and push them forward by analyzing multipath offloading for MAR among servers located at various levels of the network. We acknowledge the variety of access links and server hardwares in our study to propose a new task allocation model.

## 7.4 Modeling an AR application

In this section, we decompose a typical AR application (such as Vuforia [108]) into a set of tasks, and propose some relative deadlines and computational/networking costs for each one of them. We base ourselves on the application model proposed by Verbelen et al [44]. To this model, we add a new component – the *Feature Extractor* – to further enhance the parallelization of the application. Let us consider the concrete example of a context-aware AR web browser. Such an application analyzes the surroundings of the user, and combines geographic location and temporal informations with computation heavy image processing algorithms to display the websites of specific locations in AR. In this kind of application,

| Module | Renderer | Feature extractor | Tracker | Mapper | Object rec. |
|---|---|---|---|---|---|
| Task | $T_{r,k}$ | $T_e$ | $T_t$ | $T_m$ | $T_o$ |
| Input | Frame, Meta-data | Frame | Feature points | Feature points, World Model | Feature points |
| Output | Rendered Objects | Feature Points | Position | World Model | Object Prop. |
| $L_{data}$ | high | variable | medium | medium | medium |
| $L_{res}$ | high | variable | low | medium | low |
| Deadline | $\tau d, min$ | variable | $2\tau_{d,min}$ | $3\tau_{d,min}$ | $4\tau_{d,min}$ |
| $X$ | low | variable | medium | high | high |

Table 7.2: Tasks parameters.

image processing is the heaviest computation task, not only due to the raw computing power necessary to process a single image but also because of the frequency of the process. We design this image processing application as presented in Figure 7.1. This process starts with a video source (here, a camera) capturing pictures. These pictures then go through a *Feature Extractor* that isolates the main points of interest for future processing. These features are then fed into three interdependent components: the *Mapper* creates a dynamic map of the 3D world, the *Object Recognizer* performs fine grained analysis of the picture to locate specific objects, and the *Tracks* the objects in subsequent frames. The result of these three modules is then fed with the camera images into the *Renderer* which is in charge of combining them to display the virtual layer on top of the real world.

We break down the control flow of the application as a set of $N$ tasks $\{T_n\}$. Each task $T_n(t)$ can be characterized by its data size $L_{data,n}$, the computation results size $L_{res,n}$, the number of CPU cycles required to run the task $X_n$, and the deadline $\tau_{d,n}$, so that the overall execution time $\tau(T(t))$ is inferior to $\tau_{d,n}$. The task parameters for each components are presented Table 7.2. The *Video Source* gets the video frames from the camera. This operation requires physical access to the hardware and can only be run on the device. Nowadays, most cameras operate between 30 to 60 Frames per second, the minimum deadline is thus: $\tau_{d,min} = \frac{1}{FPS}$. At the other extremity of the pipeline, the *Renderer* aggregates the results in provenance

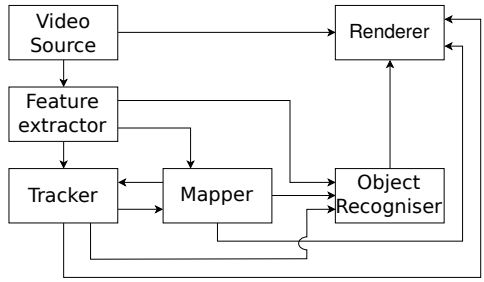



Figure 7.1: Main components of a typical MAR application.

of the computation modules and overlays them on top of the video frames. This operation has to be performed every frame, and is generally not offloaded. However, in the case of restricted hardware or heavy rendering, offloading through a low latency network may be the only solution to meet the deadline. We consider a set of k objects to be rendered in parallel, with deadline $< \tau_{d,min}$. The *Feature Extractor* extracts feature points out of the camera frames. This component can have different resolutions and deadlines, depending on the component using the feature points as input. For instance, the Tracker requires a lower resolution than the Mapper or the Object Recognizer while having shorter deadline. The *Tracker* estimates the position of the camera in the world out of feature points. This module should process 15 to 20 FPS for seamless operation, so the overall deadline for feature extraction and position estimation should be no higher than $2\tau_{d,min}$ [44]. The *Mapper* creates a model of the world out of the feature points extracted by the *Feature Extractor*. It identifies new feature points and estimates their position in space. If the required resolution is higher than for the Tracker, it is also less delay constrained and can be called every few frames. We estimate a deadline between $2\tau_{d,min}$ and $4\tau_{d,min}$ for feature extraction and world modeling. Finally, the *Object Recognizer* identifies objects out of the feature points and returns their position. Similarly to the mapper, the object recognizer does not require to be run in real time, and can be called every $4\tau_{d,min}$ to $8\tau_{d,min}$.

Out of these characteristics, we extract the dependency graph in Figure 7.2. All tasks are interdependent. Nevertheless, we can split the dependency graph for tasks to be processed in parallel [44]. As the world model and the object list do require an update every single frame, tasks $T_{e1} + T_r$, $T_{e2} + T_m$ and $T_{e3} + T_o$ can be processed in parallel. The only dependency is the combination of feature extraction with another task. By keeping track of the previous instance's results, this model avoids to pass on the excess latency to further tasks.

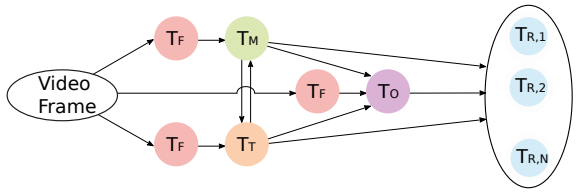



Figure 7.2: Task dependency graph.

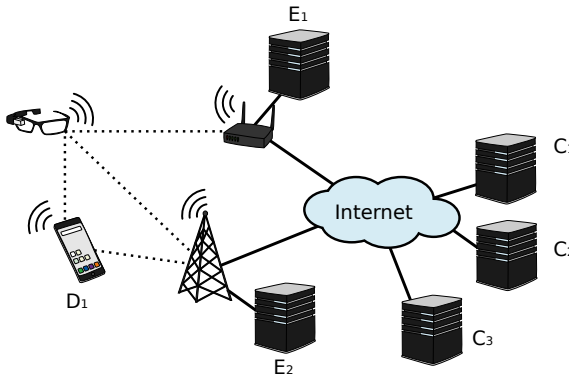



Figure 7.3: Environment model: a pair of smartglasses is connected to several computing units located at different extremities of the network: device to device ($D_k$), edge ($E_k$) and cloud ($C_k$).

## 7.5 System Model

We consider the scenario presented Figure 7.3. A mobile device (here smartglasses) executes a MAR application. Due to the low processing power of smartglasses, the computations are offloaded to companion devices, edge servers and cloud servers connected through WiFi Direct, WiFi and LTE.

### 7.5.1 Available resources

At a given time $t$, the client has a set of $N$ tasks $\{T_n(t)\}$ to run, as defined Section 7.4. The client is connected to the servers through a set of access links $\{L_i\}$. We consider that at any time $t$, only a subset of size $I$ is available to the user, depending on the actual availability of networks. Those links are characterized by their delay $\tau_i(t)$ and bandwidth $B_i(t)$, variable over time. We consider the mobile network to be down when LTE is not available, as UMTS can not provide the minimum throughput and latency requirements for MAR. The smartglasses are connected to a set of offloading devices through these networks. The set of devices includes

$J$ directly connected companion devices $\{D_j\}$, $K$ edge servers $\{E_k\}$ at the WiFi or LTE access point, and $L$ cloud servers $\{C_l\}$. They are characterized by their computing power $CPU_{\{j,k,l\}(t)}$. In the case of cloud servers, we consider the connection to be established through an aggregate link $L_{aggr,i}$ composed of one of the access links belonging to $\{L_i\}$ and a backbone network with additional latency $\tau_{backbone}$. We consider the access link as the bottleneck of the network. The resulting link $L_{aggr,i}$ is characterized by its latency $\tau_{aggr,i}(t) = \tau_i(t) + \tau_{backbone}(t)$ and bandwidth $B_{aggr,i}(t) = B_i(t)$.

### 7.5.2 Resource allocation

The execution time of task $T_k(t)$ is a function of the original transmission time $\tau_{tr}(t)$, the computation time on the server $\tau_{comp}(t)$ and $\tau_{res}$ the transmission time of the result:

$$\tau(T_n(t)) = \tau_{tr,n}(t) + \tau_{comp,n}(t + \tau_{tr}) + \tau_{res,n}(t + \tau_{tr} + \tau_{comp}) \tag{7.1}$$

with:

$$\tau_{tr,n}(t) = \begin{cases} \tau_i(t) + \dfrac{L_{data,n}}{B_i(t)} & Edge\ or\ D2D \\ \tau_i(t) + \tau_{backbone}(i) + \dfrac{L_{data,n}}{B_i(t)} & Cloud \end{cases} \tag{7.2}$$

$$\tau_{comp,n}(t) = \frac{X_n}{CPU_{j,k,l}(t)} \tag{7.3}$$

$$\tau_{res,n}(t) = \begin{cases} \tau_i(t) + \dfrac{L_{res,n}}{B_i(t)} & Edge\ or\ D2D \\ \tau_i(t) + \tau_{backbone}(i) + \dfrac{L_{res,n}}{B_i(t)} & Cloud \end{cases} \tag{7.4}$$

We consider that the smartglasses can estimate the channel conditions, as well as the available resources available on the servers at all times. Offloading $\{T_n(t)\}$ to a server comes down to assigning resources so that:

$$\tau(T_n(t)) < \tau_{d,n}\ \forall n \tag{7.5}$$

$$\min \Psi = \sum_{n \in N} \tau(T_n(t)) \tag{7.6}$$

### 7.5.3 Multipath Cloud Offloading

Cloud servers are positioned further in the network than edge servers and companion devices. As a result, transmitting over WiFi or LTE results in a lower difference in overall latency.Considering a set of access links $\{L_i\}$ connected to a backbone network to a cloud server, multipath transmission reduces overall network latency when:

$$\sum_{i \in I} \tau_i(t) + \frac{k_i}{B_i(t)} < \min_i \left( \tau_i(t) + \frac{L}{B_i(t)} \right) \tag{7.7}$$

Here, $k_i$ represents the amount of data transmitted over link $L_i$. In the most common case, only two links are present: LTE and WiFi. The system becomes:

$$\begin{aligned} \tau_{WiFi}(t) + \frac{K}{B_{WiFi}(t)} + \tau_{LTE}(t) + \frac{L-K}{B_{LTE}(t)} \\ < \min \left( \tau_{WiFi}(t) + \frac{K}{B_{WiFi}(t)}, \tau_{LTE}(t) + \frac{L-K}{B_{LTE}(t)} \right) \end{aligned} \tag{7.8}$$

with optimal K found when $\tau_{WiFi}(t) + \frac{K}{B_{WiFi}(t)} = \tau_{LTE}(t) + \frac{L-K}{B_{LTE}(t)}$:

$$K = \left( \tau_{WiFi} - \tau_{LTE} - \frac{L}{B_{LTE}} \right) \cdot \frac{B_{WiFi} B_{LTE}}{B_{LTE} - B_{WiFi}} \tag{7.9}$$

These principles can also be applied for edge servers, although edge servers display such low latency that the interconnection between the two networks may introduce a larger relative delay, reducing the impact of multipath transmission.

### 7.5.4 Mobility

Many MAR applications rely on user mobility (navigation systems, city guides, AR games etc.). The user will thus experience regular disconnections and long handover times due to his mobility. During such events, tasks offloaded to the edge will be have to be transmitted through the backbone network to be recovered through another link. We envision three scenarios: 1. If task completion prevails over latency, accept the additional delay and transmit the computation results through the backbone network. 2. In the case of latency sensitive tasks, discard the task as soon as the device leaves the access point in order not to waste resources, both at server and network level. 3. Flag critical tasks at the application level so that they are offloaded to a companion device or a cloud server in priority.

### 7.5.5 Data consistency

In traditional offloading, computation results may be kept on the server for further computation. For distributed offloading, these results must be transmitted to the rest of the system. The synchronization between a server $i$ and the rest of the system adds a delay $\tau_{sync,i}$, the time to propagate data to the system:

$$\tau_{sync,i} = \max_{j} (\tau_{tr,i \to j}) \tag{7.10}$$

### 7.5.6 Sequential processing

Several tasks may be assigned to the same link or the same server. We process tasks sequentially: only one task can be transmitted or processed at a given time on the same resource. Sequential processing permits fine-grained resource allocation as it allows prioritizing tasks according to their deadlines. Moreover, assigning a task to a given resource doesn't modify the status of already assigned tasks. Therefore:

$$\tau_{comp,n}(t) = \tau_{sched,n} + \frac{X_n}{CPU_{j,k,l}(t)} \tag{7.11}$$

$$\tau_{tr,n}(t) = \tau_{wait,n} + \tau_i(t) + \frac{L_{data,n}}{B_i(t)} \tag{7.12}$$

$\tau_{sched,n}$ being the time to wait for the task to be scheduled on the server, and $\tau_{wait,n}$ the delay before transmission on the link. If the task $T_n(t)$ has to be executed sequentially on the server, transmission of the task can be delayed by an additional $\tau_{wait,n}$ as long as $t + \tau_{tr,n} \leq t + \tau sched, n - 1$.

### 7.5.7 Tasks Dependencies

The simplest task model is the *data-partition model*, in which we only consider a pool of independent tasks at a given time. However, most AR systems cannot be decomposed into independent tasks, as most components require the output of a previous component as an input, as shown Section 7.4. Three main tasks dependencies models can be considered. For a set of $N$ interdependent tasks, the dependencies can either be linear, parallel, or a combination of both. When a set of tasks are linearly dependent, if we consider that each task result has to be reported to the smartglasses before executing another, the total execution time is:

$$\Psi = \sum_{n \in N} \tau(T_n(t)) \tag{7.13}$$

In the case of parallel dependencies where the input of task $T_N$ depends on the output of parallel tasks $T_1$ to $T_{N-1}$. The execution time of $N-1$ tasks dispatched over $N_{res}$ servers is therefore constrained by the following equation:

$$\Psi < \tau(T_1(t)) + \frac{N-2}{N_{res}} \max(\tau(T_n(t))) + \tau(T_N(t)) \tag{7.14}$$

where $n \in [2, N-1]$.

Finally, tasks can show more intricate interdependencies. This kind of topology can be resolved by aggregating parallel or linearly dependent tasks in nested clusters, with overall latency $\tau_{cluster}$, until the full system can be expressed as a linear or parallel combination of clusters.

### 7.5.8 Optimizations

Interdependent tasks introduce new constraints in the system, but also provide new opportunities for optimization. A set of $N$ linearly dependent tasks can be considered as a single task of deadline $\sum_{n \in N} \tau_{d,n}$, transmitted on the same link and executed on the same server. Execution time of this set can be reduced by transmitting all tasks sequentially and executing them as soon as they are received and the previous task completed. The overall delay for this set of tasks becomes:

$$\psi = \tau_i(t) + \sum_{n \in [1,N]} \max\left(\frac{L_{data,n}}{B_i(t)}, \tau_{comp,n-1}\right) + \tau_{comp,n} + \tau_{res,N} \tag{7.15}$$

Similarly, for parallel dependencies, all tasks may be transmitted right after task $T_1$, reducing the total time to:

$$\Psi < \tau(T_1(t)) + \frac{N-2}{N_{res}} \max_n(\tau_{comp,n}) + \max_n(\tau_{res,n}) + \tau(T_N(t)) \tag{7.16}$$

where$n \in [2, N-1]$, assuming that $\tau_{tr,n} < \tau(T_1(t)), \forall n \in [2, N-1]$.

## 7.6 Scheduling algorithm

In this section, we propose a scheduling algorithm using the model presented Section 7.5 to allocated tasks over the set of available links and servers. We first introduce a system for independent tasks, then discuss the implications of tasks with linear and parallel dependencies.

**Algorithm 1** Scheduling algorithm.

**Input:** Network bandwidth $\{B_i(t)\}$, latency $\{\tau_i(t)\}$, server available capacity $\{CPU(t)_j\}$, set of tasks $T_n(t)$, set of link/server combinations $(L_i, D_j, E_k, C_l)$;
**for** task in T.sort$\tau_d$ **do**
    **for** link, server in combination **do**
        compute $\tau_{link,server}$;
    **end for**
    allocate task $T_n$ to link/server with lowest $\tau_{link,server}$;
    remove $T_n$ from T;
**end for**
**Output:** Task allocation

### 7.6.1 Independent tasks

As described in Section 7.5.6 we consider sequential allocation for our task set, as it permits greater flexibility. We consider several metrics:

- $\{\tau_n\}$ the set of task completion time over all available servers and links.
- $\alpha_{\min} = \dfrac{\min(\tau)}{\tau_d}$.

We assume that the set of links and servers is small enough to compute $\{\tau\}$ for all tasks in a reasonable amount of time. The example shown in Figure 7.4 is representative of a typical situation, with three links, one companion device, two edges servers and three cloud servers for a total of 9 possible combinations.

We propose a simple two steps algorithm to solve the resource allocation problem. While all task have not been assigned, we compute the set of $\{\tau\}$ corresponding to all possible link/server combination for each task and allocate the task with the lowest $\alpha_{\min}$. This algorithm is further detailed summarized in Algorithm 1. This conservative algorithm aims at maximizing the amount of tasks which can be processed on time. Therefore, tasks that can be completed in the least amount of time relatively to their deadline are assigned first.

### 7.6.2 Interdependent tasks

In this section, we consider interdependent tasks, whether the dependency is linear or parallel. For more general relationships, the dependency graph can be decomposed into several linear or parallel clusters.

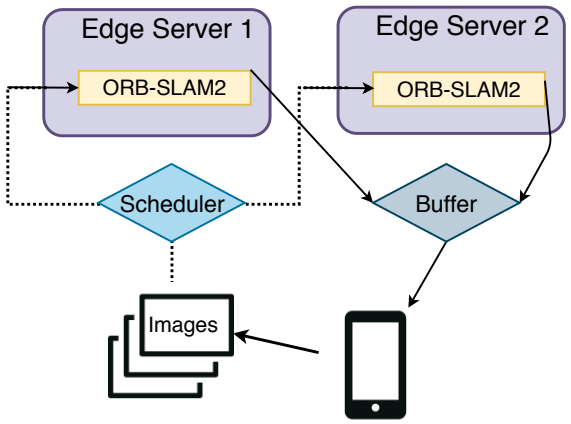



Figure 7.4: System function flow.

**Linear dependency.** A set of N linearly dependent tasks can be considered a single task of deadline $\tau_d = \tau_{d,N}$. The overal task execution time of the aggregate can be computed using the formula given in Section 7.5.8. This task can then be assigned to a single server using the aforementioned scheduling algorithm. If the tasks have independent deadlines without a constraint on the overall deadline or if we are able to estimate each deadline as a function of the overall deadline (for instance, $\tau_n = \frac{\tau_d}{X_n} \sum_{n' \in N} X_{n'}$), another strategy is to assign the first task and schedule the following tasks after it completed, recomputing the deadlines according to the actual completion time.

**Parallel Dependency.** The set of parallel tasks can be considered as a cluster of independent tasks with deadline $\tau_d = min(\{\tau_{d,n}\})$ and response time $\tau = max(\tau_n)$. Tasks in this cluster can be allocated as independent tasks, using the redefined deadline and completion time for task assignment.

## 7.7 Evaluation

In this section, we describe the implementation details and evaluation setup. We developed a showcase system which can realize real time localization offloading. The system is capable of offloading the SLAM tasks to multiple edge servers simultaneously in real time.

| Datasize | Input | Output |
|---|---|---|
| *ORB-SLAM2* | 64Kb | 12.96Kb |

Table 7.3: Data Size.

### 7.7.1 Implementation

In Linux platform, we deploy the SLAM system with `ORB-SLAM2` [109], one of the most well known real-time SLAM libraries. We implement the client device on the Android platform, of which the monocular camera captures the front-facing view. We use OpenGL to render the augmented information on top of the camera view. Our current implementation works with a monocular camera (on Android phone) but is also compatible with stereoscopic cameras.

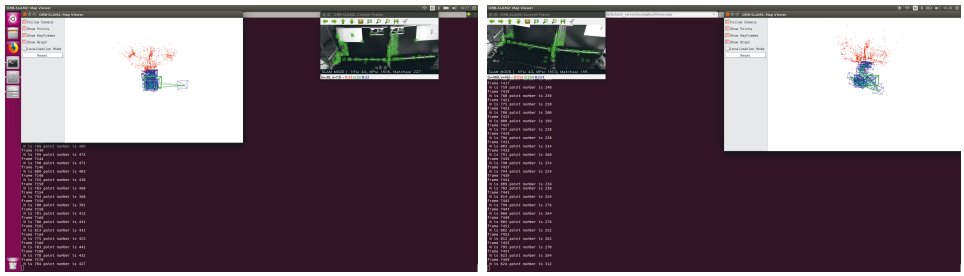
To emulate a standard mobile user, we install the client side application on a Huawei Mate9 Pro smartphone, with a 2.4 (1.8) GHz octa-core HiSilicon Kirin 960 CPU and 4GB of memory. We deploy the edge servers on two MSI GS65 Stealth 8SG[2], each of which has a 6-core I7-8750H CPU, 32GB of memory, and an Nvidia RTX 2080 Max-Q GPU. The hardware capacity of our edge server is similar to a medium-priced commodity edge server.

### 7.7.2 Scheduling VS Not Scheduling

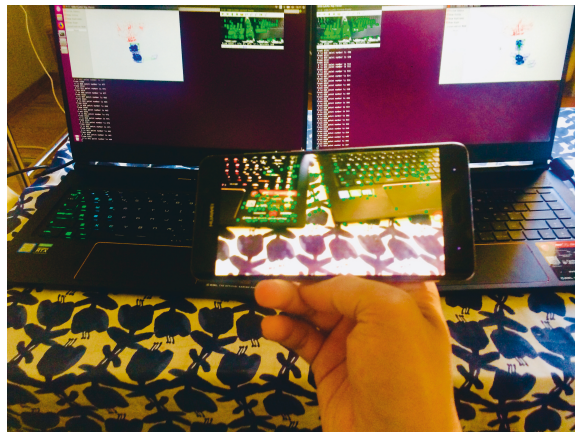
We implement a prototype of the multi-server offloading system as shown in Figure 7.5. The client phone captures images and send the gray scaled frames to edge servers via inbuilt scheduler or randomly in real time. Each edge server processes the received frame with ORB-SLAM2 and send integrated results back to the client phone via a result buffer. The client phone pulls the results from the buffer and displays in AR fashion as shown in Figure 7.5c.

The client sends compressed images at 30 fps to the programs in the server(s) via two sockets. The data size of image frame and output point cloud are shown in Table 7.3. To test the performance of scheduler under different circumstances, we throttle the uplink (client to server) to emulate bandwidth hungry scenarios. The overall uplink bandwidth is set as 3000Kb which is split between edge server 1 and 2 according to ratio 1/3, 1/4 and 1/5. Sampling 2000 received frames of the tests, we plot the feedback latency in Figure 7.6. *Schedule* and *Random* refer to the tests in which the

[2]https://www.msi.com/Laptop/GS65-Stealth-8SX/Specification

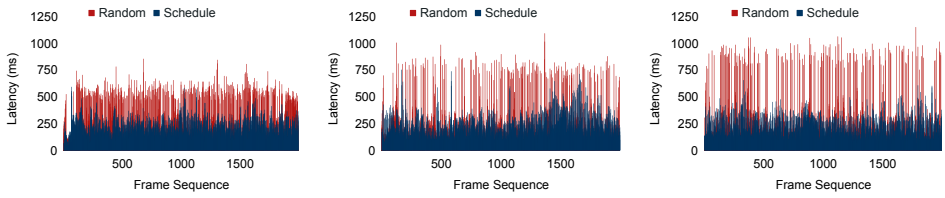


(a) Edge Server 1. (b) Edge Server 2.



(c) Mobile Client offloading to the edge servers simultaneously.

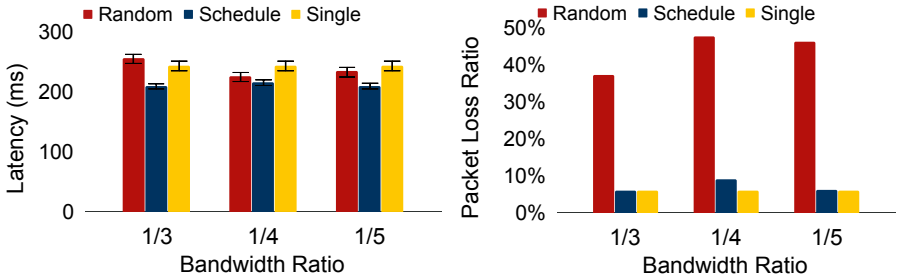
Figure 7.5: Prototype system.




(a) Bandwidth ratio 1/3. (b) Bandwidth ratio 1/4. (c) Bandwidth ratio 1/5.

Figure 7.6: Latency results.

client sends requests via scheduler or not to the two servers. We also carry out a *Single* test that the client offloads to single edge server via a uplink




(a) Latency with standard error of the mean (SEM).

(b) Packet Loss Ratio.

Figure 7.7: Result Comparison.

| Latency | 1/3 | 1/4 | 1/5 |
|---|---|---|---|
| Random | 254.844 ms | 224.961 ms | 233.766 ms |
| Schedule | 208.878 ms | 215.567 ms | 209.062 ms |
| **Packet Loss** | **1/3** | **1/4** | **1/5** |
| Random | 37.22% | 47.65% | 46.29% |
| Schedule | 6.04% | 9.06% | 6.24% |

Table 7.4: Results.

of 3000 Kb bandwidth, of which the average latency is 242.333 ms. An example of decomposition of latency is given in Figure 7.8. As summarized in Figure 7.7 and Table 7.4, multi-server with scheduling always outperforms single-server offloading and multi-server without scheduling considering the combinatory performance of feedback latency and transmission packet loss ratio.

## 7.8 Model Evaluation

To further validate the task allocation algorithms, we simulate a more complex scenario over various network conditions. Due to the lack of simulator considering both the network side and the network side, we develop our own simulator.

We use the AR application model presented Section 7.4, with the following simplifications: (1) We estimate that the Feature Extraction step

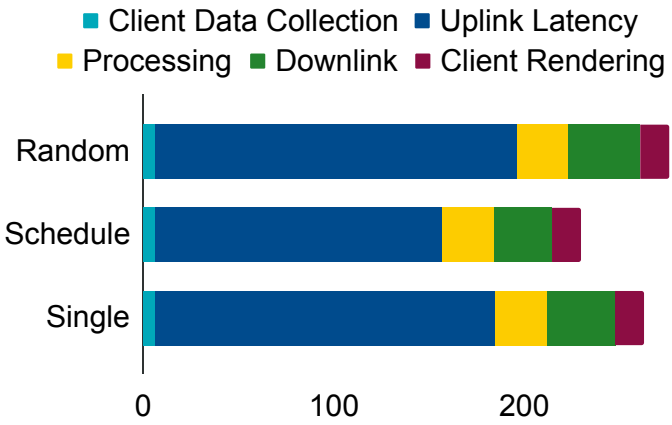



Figure 7.8: Latency decomposition for bandwidth ratio 1/3.

| | D2D | LTE | WiFi | Backbone |
|---|---|---|---|---|
| $\tau$ | 1.75ms | 9.95ms | 1.85ms | 1.15ms-19.25ms |
| $B$ | 50Mb/s | 30Mb/s | 100Mb/s | xxx |

Table 7.5: Base Network parameters.

is performed on the same server than the other tasks, and (2) We only consider a single size of JPEG frame, 64Kb, as shown in Section 7.7. As such, our model consists only of four tasks, respectively Render object $T_r$, Track Objects $T_t$, Update world $T_m$, and Recognize objects $T_o$. We use the system presented Figure 7.3 as our simulation setup. At first, we consider a network model with fixed delay and latency in order to precisely analyze the influence of the various components. Although such model does not represent a realistic vision of wireless networks, it allows us to draw valuable insight on the effect of the system's intrinsic parameters on task repartition and in-time completion. We use the measurements from Section 7.2 for our typical values. The network parameters are presented Table 7.5. We assume that the access network is the bottleneck, thus the backbone network's bandwidth is dependent equal to the access link's (either WiFi or LTE).

We define the tasks parameters as follows. First of all, we assume that $25CPU_{smartphone} < 5CPU_{edge} < CPU_{cloud}$, with $CPU_{smartphone} = 200$ and $\tau_{rendering,smartphone} = 10ms$. As such, $X_r = 2$. We then set up an Edge

| Task | $\tau_d$ | $L$ | $X$ | $L_{res}$ |
|---|---|---|---|---|
| Render object $T_r$ | 20ms | 100Kb | 2 | 100Kb |
| Track Objects $T_t$ | 60ms | 64Kb | 10 | 4Kb |
| Update world $T_m$ | 90ms | 64Kb | 26 | 12Kb |
| Recognize objects $T_o$ | 120ms | 64Kb | 15 | 4Kb |

Table 7.6: AR application tasks.

server (configuration detailed in Section 7.7) running `OpenCV`[3] for tracking, `ORB-SLAM2` [109] for mapping, and `YOLO` [110] for recognizing objects. We use 64Kb JPEG frames as input for each algorithm, and measure both the time to process the frame and the output of the algorithms. We then convert the processing time into the aforementioned complexity metric. As a result, for a 64Kb JPEG frame, $X_t = 10$, $X_m = 26$, and $X_o = 15$, and $L_{res,t} = 4\,Kb$, $L_{res,m} = 12\,Kb$, and $L_{res,o} = 4\,Kb$. Finally, we think consider that rendering the object requires not only the JPEG frame, but also the world model, the detected objects, and eventually other sensor information. We thus consider an input size of 100 Kb. We summarize these parameters in Table 7.6.
In this section, we perform the following simplifications:

- We consider that at all time t, the client has a good estimation of all the main parameters of the system: delays, bandwidth and available CPU capacity.
- We regard all those parameters as constant from the moment the task is assigned to the task completion.
- We assume that tasks are triggered by new frames.

We run the simulation over 200 s for a 30 FPS application, with rendering performed for 3 objects every 30ms, tracking every 60ms, world modeling every 90 ms and object recognition every 120ms, for a total of 24,500 tasks to allocate.

### 7.8.1 Offloading to multiple servers

First, we measure the impact of offloading to each server in the following scenarios: D2D only, edge computing only (single and multipath), cloud only (with 1 to 3 servers), and a combination of all the servers and links.

[3] `https://opencv.org/`

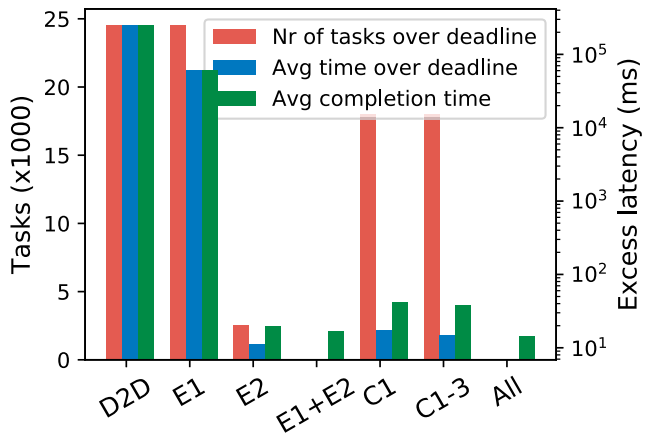


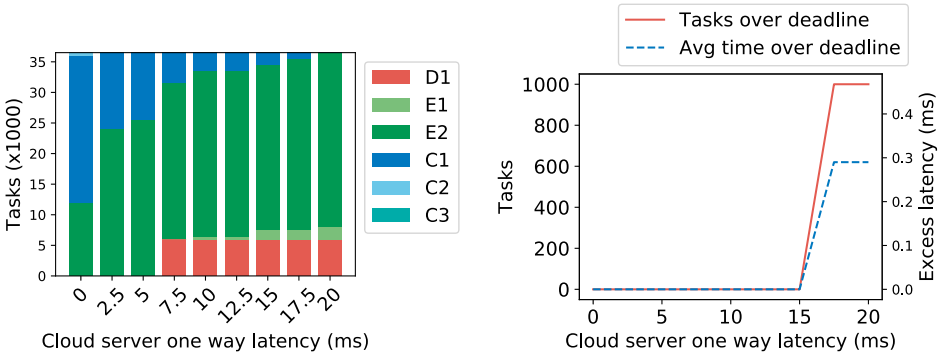

Figure 7.9: Single and multiple servers offloading. Offloading to several server lead to reasonable excess latencies (<10 ms). 100% in-time task completion is achieved by offloading to all resources in parallel.

We set the cloud latency to 7.5$ms$, representing a cloud server located in a nearby city.

Figure 7.9 shows the average completion times of tasks, the number of tasks that could not meet their deadline as well as the average excess latency for those tasks. The companion device is not powerful enough to offload the full application, excess latencies accumulate exponentially, over 250 s. Similarly, due to the high latency of the LTE link, offloading to an edge server through LTE leads to a buildup of latencies, around 60 s. Single path edge computing with WiFi shows better results, with only a small fraction of tasks completing late, and an average excess latency below 20 ms. Cloud servers, on the other hand, compensate the high network latency with the high computational power. Despite most tasks not completing on time, the average latency remains below 20 ms. Finally, transmitting to both edge servers through LTE and WiFi in parallel allows to complete all tasks on-time (avg=16.6 ms). Unsurprisingly, the combination of all elements in the system shows the best performances, allowing to complete all tasks in time with the lowest completion times (avg=14.5 ms).

### 7.8.2 Cloud distance influence

One of the main advantages of D2D or edge offloading over cloud offloading is the gain in latency from the device proximity. In this section, we wish to measure the influence of the cloud distance on the task allocation and




(a) Task assignment.

(b) Late tasks and excess latency.

Figure 7.10: Tasks allocation, late tasks, and excess latency for varying cloud distances. When the cloud latency reaches 7.5 ms, tasks are offloaded to the companion smartphone. Over 10ms, tasks are offloaded to the LTE Edge server. Over 15 ms in-time completion rate drops by 5%. The excess latency remains minimal due to task reallocation.

respect of deadlines. We consider typical one-way latency values, from 1.15 to 20 ms. We present the results in Figure 7.10.

Figure 7.10a shows the resource allocation of our algorithm depending on the cloud distance. As we can see, most of the tasks are allocated to the second edge server, through the WiFi link. Indeed, in this configuration, network delays represent the majority of overall latency. For the same reason, when the cloud latency gets higher than the LTE latency, tasks start to be offloaded first to the companion device, that presents the lowest network latency, then to the first edge server, connected through LTE. This reallocation allows to complete all tasks on-time until the one-way cloud network latency reaches 15 ms. Finally, when the cloud latency reaches 20 ms, the system entierely relies on the companion device and the edge servers. Figure 7.10b presents the number of late tasks, as well as the average time spent over the deadline. The confidence interval values were around $1e-4ms$ and could not be plotted. For $\tau_{backbone} < 15ms$, all tasks complete on time, as the simulation parameters reflect a close to ideal offloading situation. However, as soon as the $\tau_{backbone}$ reaches 15ms, 1000 tasks suddenly can not reach their deadline. A further analysis shows that most of those tasks belong to the rendering group, as they have the most constrained deadlines In this scenario, the average time spent over the

deadline is around 0.3 ms and less than 5% of tasks can not be executed within the allocated time, which remains acceptable considering the tasks as presented Figure 7.6.

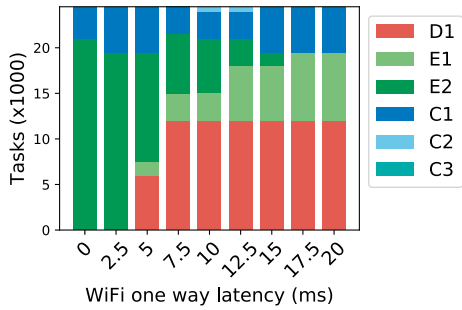
### 7.8.3 Impact of network conditions

In the previous sections, we consider network conditions close to optimal with latencies and bandwidth fixed at values measured on a university campus where both LTE and WiFi are carefully planned to cover the full area with little interference between access points. We now analyze the case of degraded WiFi performances, variable delay and bandwidth and random WiFi disconnections. In this section, we fix the backbone delay to an average value of 7.5 ms.

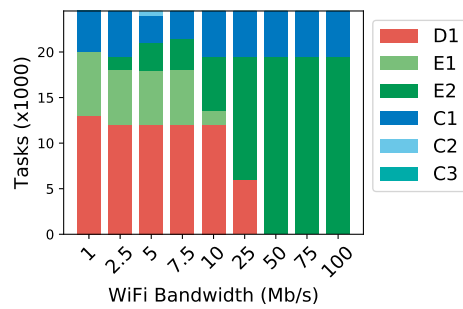
We first evaluate the effect of WiFi bandwidth and latency on the overall performance. The results are presented in Figure 7.11 Figure 7.11c presents the influence of the WiFi latency on the overall in-time task completion, and Figure 7.11a the corresponding task allocation. A slight increase in link latency immediately causes response times to exceed the deadlines. When the one way delay reaches 17.5ms, the number of late tasks as well as the average excess latency stabilizes. Indeed, the latency gets so high that the WiFi network is not considered anymore by the task allocation algorithm, and tasks get allocated to the companion device and through LTE. On the other hand, the mobile network is never used in the task allocation for $\tau_{WiFi} < \tau_{LTE}$. Inbetween, as the WiFi access link has higher bandwidth, tasks continue to get offloaded until $\tau_{WiFi} = 15ms$.

On Figure 7.11d, we represent the effect of WiFi bandwidth variation on task completion. Figure 7.11b represents the corresponding allocation. As soon as the WiFi bandwidth goes lower than the LTE bandwidth, task start not to be completed in time, although at first with minimal impact on the excess latency. Under 2.5Mb/s, the WiFi network definitely stops being used by the allocation algorithm. The link latency is therefore primordial to ensure proper execution of the offloaded application. On the other hand, the application is more tolerant to bandwidth variation, and only fails for the lowest values.

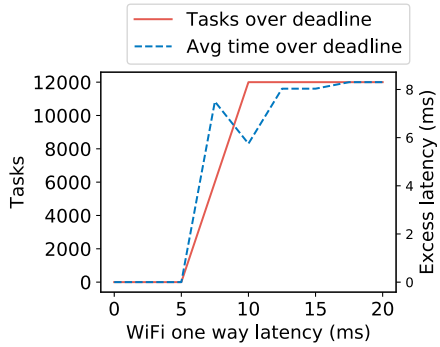
After analyzing the impact of both latency and throughput, we evaluate the effect of link instability on the task allocation. We represent the delay and bandwidth of the links as random Gaussian variables centered around the measured value presented Table 7.5. To set the standard deviation of the bandwidth and latency, we perform measurements on the university campus, located in a remote area. The WiFi measurements are performed on the Eduroam network, while the LTE measurements are done



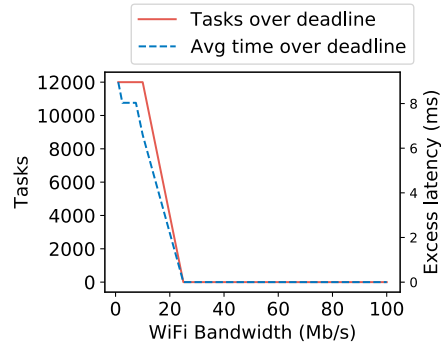

(a) Load - Variable latency.




(b) Load - Variable bandwidth.




(c) Late tasks - Variable latency.




(d) Late tasks - Variable latency.

Figure 7.11: Impact of WiFi latency and bandwidth on task allocation and excess latency. For latencies >5 ms, in-time completion rate drops to 50%. Minimum completion rate is achieved for bandwidth $> 25Mb/s$.

on a commercial operator's network. We measure the standard deviation of the latency by 100 consecutive Echo Request packets to the university server. For the WiFi link, we have an average standard deviation of 65% due to the extremely low average latency (2.7 ms one way). The latency varies between 1.7 ms and 12.1 ms For LTE, we have an average standard deviation of 13% for latency, going from 7.2ms to 13.55ms. This low variance is due to the fact that the measurements are performed in a remote area with low population density. Regarding bandwidth, we computed the standard deviation out of 10 1 Mb transfers in Iperf. We find an average standard deviation around 50% for both WiFi and LTE. We then inject these values into our model. Moreover, if LTE is generally ubiquitous, accessible WiFi networks may be sparser. We reflect this phenomenon by

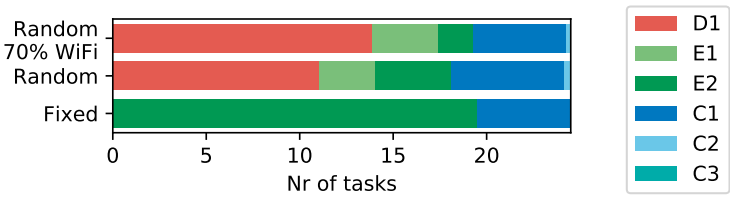


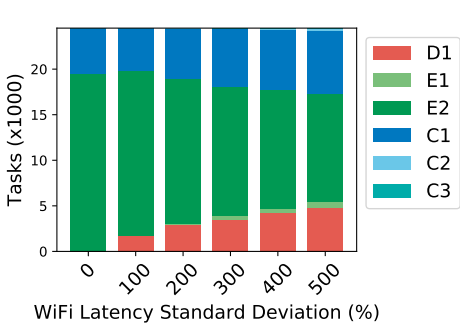

Figure 7.12: Task Allocation for Fixed channel, Random channel, and Intermittent channel.

running the simulation a second time, with WiFi networks disconnecting randomly 30% of the time.

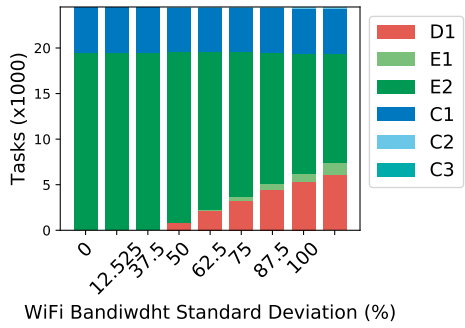
For a random channel, the overall amount of tasks exceeding their deadline remains low, less than 5%, with average excess times around 4 ms. With random WiFi disconnections, this percentage reaches 37%, with overall excess latency around 23 ms. In both cases, the task repartition, shown Figure 7.12 over the various servers is way more diverse than for fixed network conditions. The WiFi edge server is less used, while a more significant amount of tasks are allocated to the companion device. More tasks are also offloaded to the cloud servers through the LTE links. The random WiFi disconnections cause the cloud servers to be slightly less employed due to the higher latency induced by LTE usage. We notice that in both random channels and random channels with disconnections, the LTE link is used for 30% of the allocations. Indeed, when the WiFi network is not available, tasks get offloaded to the companion device, as the LTE channel displays average bandwidth and delays still too high for in-time task completion, even with higher server computing power.

As our LTE network is relatively stable, although never used in this experiment, we wish to evaluate the limits of WiFi instability on task completion rates and times. We vary the standard deviation of bandwidth from 0 to 100% (latency standard deviation fixed at 65%), and the standard deviation of latency from 50% to 500% (bandwidth standard deviation fixed at 50%). We present the results in Figure 7.13.

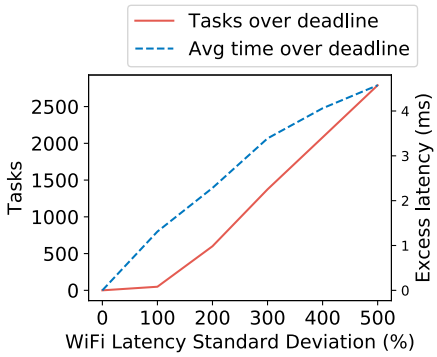
Surprisingly, we notice that WiFi latency can withstand huge variations. When the latency standard deviation reaches 100%, only 3 tasks cannot be completed on-time. On the other hand, the WiFi bandwidth can only withstand small variations, with a standard deviation up to 25% before tasks stat to complete late. However, thanks to the rest of the system, latency does not go over 5 ms for such variations.



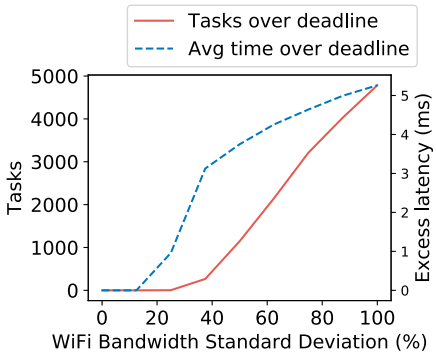

(a) Load - Variable latency.




(b) Load - Variable bandwidth.




(c) Late Tasks - Variable latency.




(d) Late Tasks - Variable bandwidth.

Figure 7.13: Impact of WiFi latency and bandwidth standard deviation on task allocation and excess latency. The WiFi latency can be highly variable before the appearance of late completion. Bandwidth is more sensitive than latency to variation, late task skyrocket for a standard deviation over 80% of the bandwidth.

### 7.8.4 Discussion

In this section, we showed that offloading to multiple servers located at various levels of the network over both D2D, WiFi and LTE links permits to overcome the limitations of single link, single server offloading. In a scenario where all tasks experience late completion in the usual paradigm, we managed to reach 100% on-time task completion. If this result can seem intuitive, our simulations show that in optimal conditions, most tasks are offloaded to a single server through a single link: the WiFi Edge server. However, the other resources relieve the system for tasks that could not complete on time otherwise, especially since excess latency adds up.

Overall, WiFi is the main communication medium, with D2D and LTE being only used as fall-back. As such, we evaluated the impact of the variation of WiFi bandwidth and latency, its randomness and its availability on the system. We noticed that if the absolute latency and bandwidth of the WiFi network are quite constrained, the system can withstand extreme variances. We also showed that even in the case of temporary unavailability, our system can use the fall-back links to provide in-time task completion (96%).

Finally, we were surprised to discover that when the WiFi network conditions cannot provide optimal task allocation, the low power companion device was preferred to the LTE edge servers. This phenomenon shows that in the case of offloading, network latency can become much more important than computation-induced latency.

## 7.9 Conclusion

In this chapter, we modeled and analyzed the impact of multipath transmission in current offloading frameworks. We first modeled the behavior of a typical AR application and integrated it within an architecture for multiple-server, multiple-link offloading. We then designed a scheduling algorithm for operating the application on the proposed architecture. We finally evaluated this algorithm through both a real-life implementation and extensive simulations. We first showed that our algorithm performs better than random allocation and than single-server offloading for equivalent aggregated bandwidth. In the simulation, we demonstrated that allocating tasks over a wide range of networks and servers noticeably improve performance over single path offloading to a single or multiple servers at the same network level. Even a very low power companion device close to the client can distinctly increase the number of tasks completed in time. Offloading over such a heterogeneous system provides robust fall-back in case some of the resources are not available. We also shed light on the dependency of the system on network latency. However, the system was surprisingly resilient to random variations, both in terms of bandwidth and latency. One of the main limitations is the maintenance cost: the system assumes that the developer can maintain a full cloud and edge infrastructure, and the user does restrict his mobile data usage. However, we showed that the LTE network is mainly a fall-back network, and WiFi is the default network for offloading.

In the future work after the thesis, we plan to further evaluate the imbrication of the different parameters of the system and extend our real-life implementation. We will focus on other types of optimizations that can happen further in the network, such as task aggregation and chaining. Finally, we will study the impact of 5G, which announced bandwidth and latency will lead to a higher utilization of the mobile link.

# Chapter 8

# Intelligent Traffic Control

## 8.1 Overview

Traffic congestion is worsening in every major city and brings increasing costs to governments and drivers. Vehicular networks provide the ability to collect more data from vehicles and roadside units, and sense traffic in real time. They represent a promising solution to alleviate traffic jams in urban environments. However, while the collected information is valuable, an efficient solution for better and faster utilization to alleviate congestion has yet to be developed. Current solutions are either based on mathematical models, which do not account for complex traffic scenarios or small-scale machine learning algorithms. In this chapter, we propose ERL, a solution based on Edge Computing nodes to collect traffic data. ERL alleviates congestion by providing intelligent optimized traffic light control in real time. Edge servers run fast reinforcement learning algorithms to tune the metrics of the traffic signal control algorithm ran for each intersection. ERL operates within the coverage area of the edge server, and uses aggregated data from neighboring edge servers to provide city-scale congestion control. The evaluation based on real map data shows that our system decreases 48.71% average waiting time and 32.77% trip duration in normally congested areas, with very fast training in ordinary servers.

## 8.2 Introduction

Traffic congestion is continuously rising in most urban areas around the world. Multiple factors contribute to this situation, among which the increasing number of vehicles, the inadequate infrastructure, and the distribution of points of interests around the city. Traffic congestion affects the

environment, the individual wellbeing of citizens, and has a considerable impact on the economy. In 2017, drivers spent on average 102 peak hours in congestion in Los Angeles, 91 in Moscow and New York City, 74 in London, and 69 in Paris [111]. Current solutions either rely on traffic center's control or on centralized cloud services such as Google map navigation. However, the continuous increase in urban congestion shows that these solutions need drastic improvement.

Traffic congestion includes a predictable part caused by traffic increase and a non-negligible unpredictable part caused by incidents. Several models already account for the predictable part of congestion. However, to the best of our knowledge, few studies focus on alleviating traffic congestion caused by accidents and incidents. Moreover, although predictable traffic congestion can be solved through long-term infrastructure investments, the consequences of incidents can only be handled in real-time. For these reasons, the best solution to mitigate the impact of traffic congestion is to empower the signal timing plans with traffic-responsive capabilities while notifying incidents and rerouting neighbor vehicles.

In this chapter, we propose ERL, an integrated framework to optimize traffic signals and keep drivers updated with the newest traffic conditions. ERL divides the urban traffic into areas defined at intersection, neighborhood and district level. Each level is optimized by a "local edge" device coordinating the edge servers at lower levels. This modular design provides fine-grained traffic optimization while considering the whole city traffic in real time. As the three levels of optimization run in parallel, a local area can optimize its traffic signal plan quickly without waiting for city-scale orchestration. On the other hand, ERL also allows reorganizing the traffic at a larger scale in the case of massive congestion.

ERL's architecture is massively distributed and relies on a pervasive deployment of edge servers, including local monitors at intersections and edge servers at base stations and aggregation points. By focusing on a given small-scale area, ERL allows to considerably simplify the algorithms implemented in each server. For instance, the machine algorithms employed within edge servers are specifically trained for a given area, which limits the possible outcomes. This structure allows for the deployment of specialized lightweight edgelets that require minimal computing capabilities at each level.

The contributions of this chapter are as follows: 1. *Design of ERL*, an integrated framework for handling traffic congestion in real-time at city-scale, 2. *Usage of Reinforcement Learning (RL) algorithms to automatically control the traffic signals* depending on the ingoing traffic. To the best of

our knowledge, ERL is the first reinforcement learning proposal to optimize traffic signals on neighborhood scale (see Section 8.3), 3. *Extensive simulation* based on real-world data to evaluate the traffic improvement brought by ERL.

The rest of chapter is structured as follows. We cover related work in Section 8.3. In Section 8.4 we present the overall system design and traffic model. We give practical details of our algorithm in Section 8.5 and present our evaluation in Section 8.6.

## 8.3 related work

Researchers have put a lot of effort into optimizing traffic using data analysis based on auxiliary instruments and techniques [112]. Most studies either adapt traffic lights or encourage drivers to take better decisions to alleviate congestion [113–115]. ERL focuses on adapting traffic lights while providing the required fast interactive controls required for rerouting drivers.

**Artificial intelligence:** Most related works in this area nowadays adopt reinforcement learning algorithm with different adaptations for specific goals. However, the action to take mostly falls in two approaches: turning on/off the light directly [116–118] or change the light phase directly [119] based on trained DNN. The obvious disadvantage of the former approach is that immediately transitioning from the current traffic signal phase to the selected action can cause incidents. Though authors in [117] proposed to add additional traffic signal phase configurations preceding the chosen action, different intersections have varied traffic light phase group which require a number of specific configurations. The same issue exists for the latter approach, which also requires additional effort [119].

Moreover, these approaches require a large amount of data to scale to multiple traffic lights,because they have to determine the action for each intersection's lights explicitly. For instance, a typical intersection can easily have more than 6 phases and the training for 15 intersections requires more than 470 GB memory space. Albeit similarly, neighboring lights still have differences in their traffic conditions, preventing coupling to improve the overall performance. As such, local decisions without awareness of neighborhood congestion can hurt traffic control performance by disturbing any inherent city-scale traffic balance through greedy local decisions.

Our proposal, on the other hand, adapt traffic lights indirectly by tuning the thresholds of the phase control algorithm in each intersection. With this design, ERL tunes the **sensitivities** of traffic lights instead of changing them directly. As such, adapting the sensitivities of neighboring traffic

lights as a group is reasonable since they experience similar traffic conditions, while leaving each intersection some freedom to decide its light phase depending on its own traffic condition. ERL has much better scalability and faster DNN training.

## 8.4 System Design

In this section, we introduce the architecture of ERL and describe the major communication processes. To provide a comprehensive understanding of the system function, we also include the processes of V2E (Vehicle to Edge) which is out of the scope of this work (refer to our previous work [76]).

### 8.4.1 System Architecture

ERL involves around two key layers: the **device layer** and the **edge layer**. The **device layer** includes the vehicles, roadside buildings, infrastructures, traffic signals, intersection monitors, and any other devices involved in the vehicular network. In the rest of this chapter, we assume there is at least one device monitoring each intersection. This device embeds video cameras facing all directions and is capable of wireless communication. It therefore captures and transmits all nearby traffic data. We assume the monitor and traffic lights at each intersection are co-located and can transfer data to each other freely and instantly. We call "client" any object in the device layer that transfers data to the ES. The **edge layer** host the ESes at the core of our architecture. We distribute these ESes hierarchically in two tiers. The first tier (T1 ES) consists of ESes co-located with the base stations at the access network level[1]. Second tier ESes (T2 ES) are co-located with aggregation points in the core network. Our placement scheme provide ESes with faster awareness of traffic condition, congestion, and incident, while minimizing the average distance of vehicle to edge and deployment cost. Finally, a remote cloud may provide on-demand backup and aggregation capabilities, which are not our major concern in this work. Please refer to [76] for more details and communication processes.

### 8.4.2 Traffic Model

Due to space limitation, we briefly describe our traffic model. We formulate our model based on a macroscopic model [120] that yields a sufficiently accurate description of the changing traffic flows in large areas (e.g. a

[1]Base station in this paper refers to the entity at the edge of the fixed network, e.g., BTS, eNB, and gNB.

central urban area), given the road network, the traffic load and the traffic conditions. Each ES controls the traffic lights in its covered area. To avoid overlapped control demands from different ESes, the assignment of traffic lights to ESes is predefined by the system. Let us assume there are $\mathcal{E}$ T1 ESes in the road network, each of which covers an area consisting of a unidirectional link set $\mathcal{L}^e = \{\mathcal{L}_i | i = 1, ..., I^e\}$ $(e \in \{1, ..., \mathcal{E}\})$. We use $e$ to indicate both the index of ES and its covered area in the rest of this chapter. Each link represents a lane connecting two subsequent intersections. Lane changing is not considered.

We assume the traffic lights in the urban area pertain to a common signal timing plan, characterized by a fixed cycle containing $\mathcal{F}$ phases. A phase refers to the time duration of the green lights for a given direction. Let $n_i(k)$, $N_i$, $I_{int}$ respectively represent the number of vehicles in $L_i$ at the beginning of the $k$th cycle, the capacity of $L_i$, and the indices of the internal links inside an ES-covered area. The problem can be formulated as follows:

$$\min_{t^f(k)} N_e(k) = \min_{t^f(k)} \frac{1}{N} \left[ \sum_{k=1}^{N} \sum_{i \in I_{int}} n_i(k) \right] \tag{8.1}$$

for internal ES-covered area traffic optimization, where $t^f(k)$ is the duration of phase fin the kth cycle, $N\_e(k)$ is the total number of vehicles in the area covered by ES $e$.

## 8.5 Algorithm

The optimization process includes three parallel and interactive threads on three distinct levels, i.e., *intersection* level, *intra ES covered area* level and *inter ES covered area* level, respectively performed by traffic lights, T1 ESes and T2 ESes. The basic optimization of traffic is performed by the individual sets of traffic lights at each intersection. Each traffic light set adapts its red and green light phases to minimize the waiting queue at the intersection, thanks to phase adaptation algorithms. At a higher layer, T1 ESes optimize the traffic in their covered areas, by tuning the metrics of the traffic light control algorithm across the intersections in the area. We base the tuning of metrics on reinforcement learning. T2 ESes optimize the urban traffic, by tuning the degree of optimization of each T1 ES according to the traffic density.

**Intersection level**: traffic lights are able to capture the length of jam thanks to their embedded cameras, and adapt the duration of phases based

on a traffic light control algorithm. This algorithm relies on the following parameters: the *decision time interval* (time it takes to decide an adaptation) $\mathbf{t_{decide}}$, the *decision threshold* **TH** and the *looking distance* [121,122]. For each intersection, the traffic lights have an initial signal cycle $T_c$, consisting of the phases of green lights in four directions, respectively $t_E$, $t_e$, $t_N$ and $t_n$, $t_E$, $t_e$, $t_N$ and $t_n$ We consider two *monitoring metrics* on intersection level light adaptation: the average vehicle speed and waiting queue length. $M_E$, $M_e$,$M_N$ and $M_n$ indicate the number of vehicles waiting in the aforementioned directions. After $t_{decide}$, if the ratio of the monitoring metric is larger than the threshold, the system extends the phase of green light for the direction with worse performance. Meanwhile, it decreases the equivalent length of the green phase for the opposite direction. We consider $M_x^{ratio} = (S_y - S_x/S_y)$ to dictate the ratio of average speed in direction $x$, $M_x^{ratio} = (W_x - W_y/W_x)$ to dictate the ratio of waiting queue length in direction $x$, where $y$ indicate the opponent direction of $x$.

**Intra area optimization**: T1 ESes carry out internal traffic optimization by tuning the metrics of traffic light algorithms at the intersections in their coverage area. For simplicity, we only consider the threshold metric $TH$ of the intersection level algorithm to be tuned. With I2E (infrastructure to edge) communication, T1 ESes collect the monitoring data of their covered area in close to real-time and tune the traffic lights according to Algorithm 2. The optimization uses reinforcement learning algorithm, based on a deep neural network to learn the optimal traffic signal control metric. We adopted deep Q-learning to provide an adaptive algorithm responding to dynamically changing traffic condition. The advantage of Q-learning for traffic signal control is described in more details in a study by Abdulhai et al. [123]. Next, we define the intra area state $S_t$, agent action $A_t$ and reward $R_t$.

*Intra Area State:* ES **e** needs the following intra area information to tune the metrics of traffic signal control algorithm: traffic flow, average vehicle speed at each road and signal control algorithm state. To represent the traffic flow and the vehicle speed, we collect the number of vehicles in each lane and their average speed into a matrix of traffic flow $\mathbf{F_e}$ and a matrix of average vehicle speed $\overline{\mathbf{V}}_\mathbf{e}$. The number of vehicles in each lane is normalized by lane capacity and recorded at the corresponding entry of matrix $\overline{\mathbf{V}}_\mathbf{e}$. In the following equations, $A^\intercal$ represents the transposition of matrix $A$. The matrix $\mathbf{F_e}$ of traffic flow for all roads in the area is defined as follows:

$$F = [n_1(t), n_2(t), \ldots, n_{I^e}(t)]^\intercal \tag{8.2}$$

where $n_i(t)$ is the number of vehicles in lane $L_i$ at time $t$, $I^e$ is the number

of lanes in the area covered by $e$. Similarly, the matrix of average speed for all roads in the area is:

$$V = [\overline{v}_1(t), \overline{v}_2(t), \ldots, \overline{v}_{I^e}(t)]^\intercal \tag{8.3}$$

We use a vector **TH** to represent the threshold metrics of the signal control algorithm of each intersection.

$$TH = [\overline{th}_1(t), \overline{th}_2(t), \ldots, \overline{th}_{I^e}(t)]^\intercal \tag{8.4}$$

where $n_i(t)$ is the number of vehicles in lane $L_i$ at time $t$, $l^e$ is the number of traffic lights in the area covered by $e$. At the beginning of time step t, the agent observes intra area state $S_t = (\mathbf{F}, \mathbf{V}, \mathbf{TH})$ for intra area traffic control.

*Agent Action:* After observing intra area state $S_t$ at the beginning of time step $t$, the agent chooses one action $A_t \in \{-1, 0, 1\}$: decreasing, maintaining or increasing the value of threshold metric $th$ for each traffic light. The threshold $TH$ can be seen as the sensitivity of the light control algorithm: the algorithm changes the green light phase more frequently with a lower threshold. For instance, in a central area, the traffic flow varies a lot from morning to evening, which requires a higher sensitivity to adjust the signals. Rural areas, on the other hand, can have small traffic flow all the time, which can be satisfied with "retarded" light control.

*Reward:* The reward is the change of the speed and number of vehicles between two neighboring cycles as follows:

$$R_t = A * (V_{t+1} - V_t) + B * (N_t - N_{t+1}) \tag{8.5}$$

where $A$ and $B$ are weight metrics.

*Agent Goal:* The overall goal of ERL is to optimize the traffic control. As a result, ERL makes the traffic flow smoother, with shorter waiting times and higher average speeds in the long run. The agent needs to find an action policy $\pi*$ that maximizes the cumulative future reward as follows (Q-value):

$$Q^\pi(s, a) = \exp\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi\right] \tag{8.6}$$

where $\gamma$ is a discount parameter, $0 \leq \gamma \leq 1$, reflecting the weight the agent puts on future rewards. To put it another way, the goal is to find following $\pi*$:

$$\pi* = \arg\max_{\pi} Q_\pi(s, a) \tag{8.7}$$

*Deep Neural Network (DNN):* We use a simplified neural network architecture. The network takes the state vector $S_t$ as the input, following

two hidden fully connected layers of 2000 and then 3000 neurons with rectifier nonlinear activation functions. The output layer is $|A_t|^{l^e}$ ($l^e$ is the number of traffic lights) neurons with linear activation functions.

To decrease the output layer dimension, we propose to *combine the threshold adaptation of neighbor traffic lights. Since our agent action is to tune the threshold value of the intersection level algorithm, combining the threshold adaptation of neighbor traffic lights does not couple their phase adaptations.* To put it in a simpler way, we only couple the sensitivity of neighbor traffic lights. Their phase changes depend on the real traffic. We train the network using ADAptive Moment estimation (Adam) [124] which fits for fast convergence and satisfactory performance [125]. We adopt the $\epsilon$-greedy method to let the agent selects the action with the current biggest estimated $Q$-value with probability $1 - \epsilon$ and randomly selects one action with $\epsilon$ at each time period (see Algorithm 2). To reduce cost, we adopt minibatch method. The agent randomly draws a batch of samples from the replay memory M to form input data and target pairs and update DNN weights $\theta$ by Adam algorithm. In Section 8.6, we test different action periods, each of which corresponds to a feasible minibatch size.

**Inter area optimization**: Each ES coverage area can be considered as a "big intersection" in which the internal traffic flow is optimized with *intra area optimization.* Since each ES is placed in the cluster center based on traffic density, the traffic flow between the ES coverage areas is more likely to be lower. However, if ES areas are overlapping, the optimization of a congested area may affect the other areas. That is to say, if an ES relieves the congestion inside its covered area, output links will have higher traffic flow towards neighboring areas. These areas will then see their internal traffic flow increase. For this reason, Tier 2 ESes optimize the inter area traffic following the same methodology with the intersection level algorithm, that is, slowing down the optimization of a T1 ES area if its impact on neighboring areas outdoes the improvement of its internal traffic. In this chapter, we focus on evaluating the first two optimizations, and leave the evaluation for inter area optimization for future work.

The motivation behind this design is threefold: (i) Light control on intersection level should be in real time, that is, phase decision should be made by a simple algorithm. Threshold-based algorithms fulfills this requirement. (ii) Though being fast, threshold-based algorithms lack the ability to optimize traffic light control facing different traffic conditions and surrounded road maps. The reinforcement learning algorithm tunes the thresholds in different intersections to provide comprehensive optimization. (iii)The distribution of ERL's computational complexity fits the nature of

**Algorithm 2** Intra ES area algorithm

Initialize DNN network with random weights $\theta$;
Initialize $\epsilon$,$\gamma$,$N$;
**for** epoch = 1 to $N$ **do**
    Initialize intra ES area state $S_1$;
    Initialize action $A_0$;
    Start new time step;
    **for** time = 1 to $T$ seconds **do**
        Based on observed state $S_t$,
        the agent selects action $A_t = \arg\max_a Q(S_t, a; \theta)$ with probability $1-\epsilon$ and randomly selects an action $A_t$ with probability $\epsilon$;
        **if** $A_t == A_{t-1}$ **then**
            keep thresholds unchanged
        **else**
            send new thresholds to the traffic lights in covered intersections according to $A_t$
        **end if**
        Increment simulation by period $t$
        The agent observes reward $R_t$ and next state $S_{t+1}$;
        Store observed experience $(S_t, A_t, R_t, S_{t+1})$ into replay memory $M$;
    **end for**
    Randomly draw minibatch samples $(S_i, A_i, R_i, S_{i+1})$ from $M$
    Batch training: input state and targets and train the network according to Eq. 8.6 and Eq. 8.7.
**end for**

infrastructure deployment, which is, intersection light with limited resource works better with a simpler algorithm, which edge server is capable of much heavier computation. Yet, for fast computation and due to resource upper bound, it is reasonable to assign local traffic light control to an edge server instead of a larger scale task.

## 8.6 Simulation

We deployed an ES on a Linux server, with a single core Intel(R) Xeon(R) CPU E5-2680 v4, 10GB of memory partition, and an NVIDIA Tesla P100 GPU. The hardware specification of our ES is similar to a cheap-priced edge server in 2018 [78]. As such, we test the system performance with-

out relying on expensive hardware. We build our reinforcement learning algorithm with Keras [126] and run the tests on the Simulation of Urban MObility (SUMO) [127] simulator.

### 8.6.1 Simulation Settings

To extensively test our system, we focus on different metrics that may influence the performance. The overall goal of the simulations is to compare the system performance in different scenarios, find out the best metrics, and suitable scenarios.

**Map:** We evaluate our system on real map data extracted from Open Street Map (OSM) [128]. We select different scales centering on Times Square, New York City, which is one of the most congested area in the world. This area contains a dense distribution of road and traffic lights, which fits our goal of testing the system on traffic light control. Moreover, it gives a lower bound of system performance with a large proportion of unidirectional roads and not-four-direction traffic lights. We test on two areas containing 67 and 127 traffic lights.

**Traffic:** We generate the traffic with a random algorithm, that a specific amount number of vehicles are generated per hour/lane-kilometer, each of which has a random departure and destination road. We test on 100, 600 and 1200 vehicles per hour/lane-kilometer. Taking 100V (vehicle) as an example, it means that 400 vehicles will be generated on a 4-lane unidirectional road per hour. Up-to-date public New York traffic data is out of reach. Therefore, to map the generated traffic to real life, we compare it with the traffic data in London central area[2]. We find that $100V/h/lane-km$ is comparable with the peak time traffic in central London (only with similar length roads). Meanwhile, $600V/h/lane-km$ and $1200V/h/lane-km$ are paranormal traffic volume which we only use to test the system performance in extreme scenarios.

**Monitoring metric:** As introduced in Section 8.5, each intersection can tune its traffic lights using the intersection level algorithm based on average vehicle speed or waiting queue length. To extend the measurement to cover the whole roads, we use the number of vehicles with speed of less than $0.1m/s$ ($halting$) on each lane instead of waiting queue length. We test and compare the two metrics in our evaluation.

**Adaption period:** As mentioned in Section 8.5, we test different adaptation period, i.e., $100s$, $300s$, $600s$, $800s$ and $1000s$ to find out the best period for adapting the threshold of the intersection level algorithm. We

[2] `https://data.gov.uk/dataset/gb-road-traffic-counts`

set the test interval as such so the lights phases can be tuned upon a appropriate period.

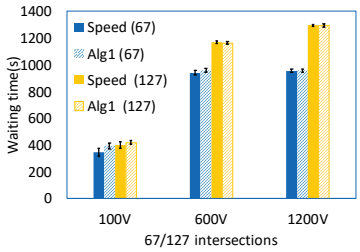
**Adaption unit:** We also test different adaptation unit, i.e., how much a change being made to a threshold value in each action. Unlike other works, we do not assume a specific threshold works best for all scenarios. Here we test on different adaptation units of *default value, increment, decrement*, i.e., 0.2/0.2/0.2, 0.2/0.2/0.05 and 0.1/0.1/0.1.

### 8.6.2 Simulation Results

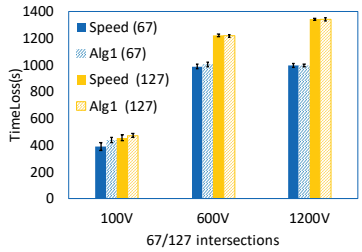
We evaluate the performance based on the following metrics: average waiting time (time in which the vehicle speed was below $0.1m/s$), average time loss (time lost due to driving below the ideal speed), average depart delay (time the vehicle had to wait before it could start his journey due to lack of road space), and average trip duration. Unless stated differently, all of our tests are one-hour long and we use the approach with adaptation period *300s*, adaptation unit 0.2/0.2/0.2, monitoring metric *average speed* by **default**. And in all the tests, we use *the intersection level algorithm* (without Algorithm 2) as a comparison approach.

First, Figure 8.1 to Figure 8.4 all show that our system improves all the performance metrics in all the scenarios of $100V/h/lane - km$. On the other hand, in the two extreme scenarios of 600 and 1200 $V/h/lane - km$, our system provides very limited help, even minor controversial impact in some cases (Figure 8.1d). This shows that our system can alleviate traffic congestion in normal scenarios. However, in extremely bad scenarios (600V and 1200V/h/lane-km), the traffic is so congested that there is nothing much we can do to help. For instance, the average waiting time reaches 1200s in 127-traffic-light scenarios (Figure 8.1a), which is about 75% of the average trip duration. This is obviously beyond the scope that traffic light control can help with.

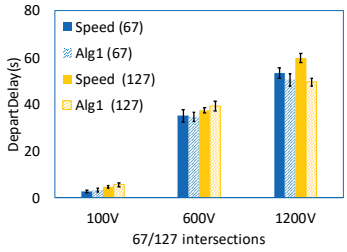
Figure 8.1 shows that ERL can provide considerable improvement in 67-light map. In 127-light map, the improvement is much smaller. This shows that our work fits small scale maps which aligns with our expectation. Because we wanted to realize fast control in distributed edge servers, each of which covers a small area. Only within a small area, the amount of collected data allows fast DNN training and tuning. Cloud collects data at a larger scale but provides much slower DNN training. As such, cloud service certainly cannot provide DNN training and control feedback as fast as ERL. Figure 8.2 shows that monitoring the number of halting vehicles provides better performance than average speed. Figure 8.3 and Figure 8.4 show that approach unit 0.2/0.2/0.2 and period 800s outperform others.



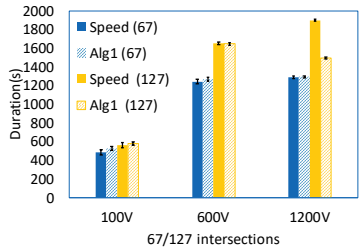

(a) Average waiting time.




(b) Average time loss.

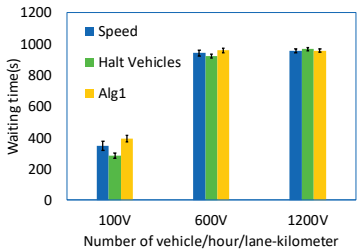



(c) Average depart delay.

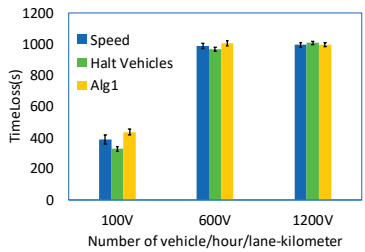



(d) Average trip duration.

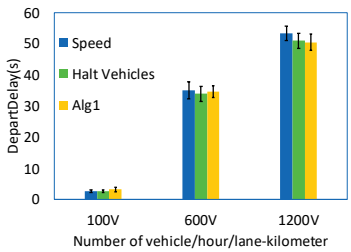
Figure 8.1: Statistics on different scales.



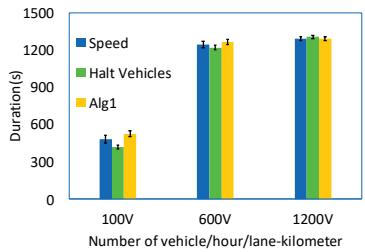

(a) Average waiting time.




(b) Average time loss.

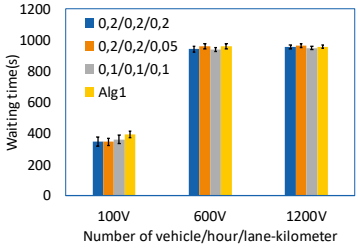



(c) Average depart delay.

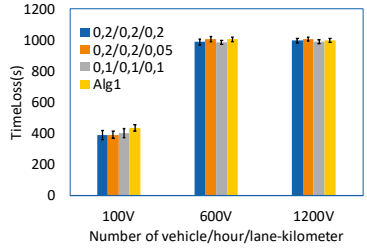



(d) Average trip duration.

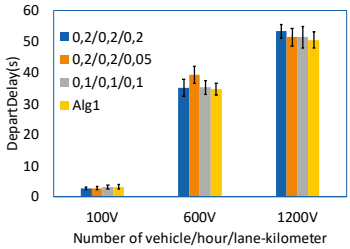
Figure 8.2: Statistics on different monitors.



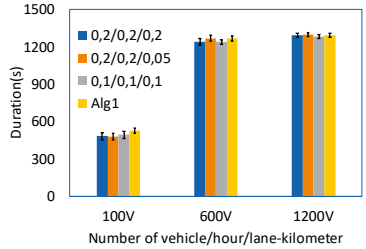

(a) Average waiting time.




(b) Average time loss.

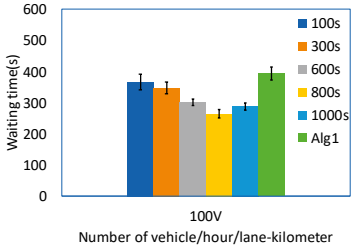



(c) Average depart delay.

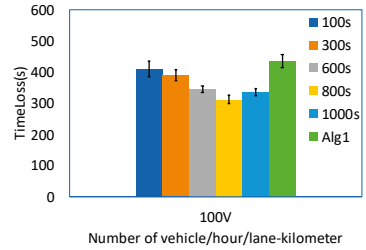



(d) Average trip duration.

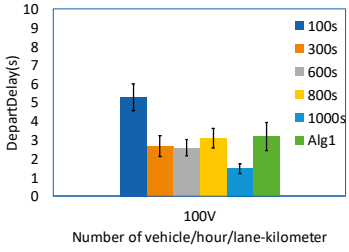
Figure 8.3: Statistics on different units.



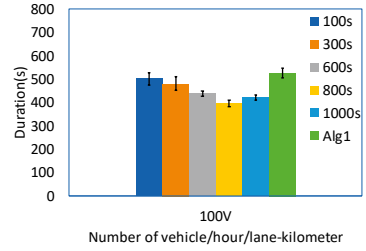

(a) Average waiting time.




(b) Average time loss.




(c) Average depart delay.




(d) Average trip duration.

Figure 8.4: Statistics on different periods.

As a summary, ERL trains DNN based on **one-hour** traffic data within 9m30s on average. Therefore within every 10 minutes, ERL is able to update the DNN based on previous hourly traffic data. The improvement in our evaluation includes decreasing at most 48.71% average waiting time, 39.49% time loss, 3.12% depart delay and 32.77% trip duration (Figure 8.4), on the map of central New York area that contains 67 traffic lights with normal congestion level (100V/h/lane-km).

## 8.7 Conclusion

In this chapter, we present ERL, an architectural framework for traffic lights optimization. Our system exploits the low latency of Edge servers to provide fast DNN training and control feedback. Thanks to its layered architecture and algorithm, ERL runs optimization at intersection, neighborhood and city level that allow for different fine-grained and scale of traffic control. As a first step, we evaluate the first two layers of optimization in this work. Requiring only ordinary hardware, ERL can decrease 48.71% average waiting time at normal congestion scenario. Unlike other works, we propose the architectural algorithm and select threshold of phase control as the action target. With this indirect control methodology, we enable the coupling of neighboring lights adaptation and decrease the dimension of action space allowing ERL to scale to city block size with fast training and control feedback.

As a first step, we explore the performance of the algorithm on intra ES area level. In the future work after the thesis, we will focus on improving the intra ES area algorithm and evaluate inter ES traffic optimization.

# Chapter 9

# Conclusion

This thesis has investigated edge-facilitated mobile computing and communication systems. This chapter summarises the contributions and research results of this thesis and outlines potential future directions for the work.

## 9.1 Summary

In Chapter 3 and Chapter 4, we have studied the utilization efficiency of edge computing resource. We considered edge caching as an example and investigated various grouping strategies with a range of workloads and parameters. The results have consistently shown that when user interest profiles are different from each other, grouping users of one interest profile into one edge cache yields considerable benefits in terms of overall cache performance. We also have developed a solution to predict and store data in edge resource caches for upcoming computations based on existing edge and fog computing models. Targeting applications include industrial environment, particularly in factory automation and collaborative robotics. The simulation results have shown that grouping based on workload type significantly improved system performance in edge clouds through reduced network traffic and access latency.

In Chapter 5, we have proposed a novel Vehicle-to-Edge system for AR applications in vehicle-to-everything (V2X). We chose Connected Vehicle Vision as the representative use case. Starting from potential exploration, we showed the promising capacity of edge computing to facilitate this type of use cases. Thenceforth we presented corresponding solutions in details including edge server deployment and placement, edge service functionality design, data flow and communication mechanisms. To validate the concepts behind the system, we built a prototype application that we evaluated

through real-world indoor experiments. We also did a scalability test with datasets of London and primarily showed that our system can improve the performance of use case with reasonable cost.

Chapter 7 provided an extension to current mobile edge offloading models using multiple paths specified for MAR applications. We presented a model for multi-server device-to-device, edge and cloud offloading. We first modeled the behavior of a typical AR application and integrated it within an architecture for multiple-server, multiple-link offloading. We then designed a scheduling algorithm for operating the application on the proposed architecture. We evaluated the proposal through both a real-life implementation and extensive simulations. The results have shown that our algorithm performs better than random allocation and than single-server offloading for equivalent aggregated bandwidth. We have also demonstrated that allocating tasks over a wide range of networks and servers noticeably improve performance over single path offloading to a single or multiple servers at the same network level.

Finally, in Chapter 8 we looked into the potential of combining the technology of edge computing and machine learning. It has proposed ERL, a solution that exploits the low latency of Edge servers to provide fast traffic data collection and DNN training to alleviate congestion by providing intelligent optimized traffic light control in real time. Thanks to its layered architecture and algorithm, ERL runs optimization at intersection, neighborhood and city level that allow for different fine-grained and scale of traffic control. We have evaluated the first two layers of optimization and shown that ERL can decrease 48.71% average waiting time at normal congestion scenario requiring only ordinary hardware. We proposed the architectural algorithm and select threshold of phase control as the action target to enable the coupling of neighboring lights adaptation and decrease the dimension of action space. This allows ERL to scale to city block size with fast training and control feedback.

## 9.2 Future Directions

This thesis can be extended in multiple directions. Each chapter outlines its own future directions, of which some are related to or overlapped with others. The field of edge-facilitated mobile computing and communication holds much promises for future research, especially for bandwidth-hungry and latency-sensitive applications. Below we describe the detailed potentialities of this field for future extensions.

This thesis considers the major trend of edge computing architecture nowadays in the sense that edge servers and mobile clients are set up separately. However, as the computational capabilities of mobile client devices grow rapidly, future edge computing solution should consider to expand its resource pool to involve the client devices that are willing to share their resource freely or with payment. For instance, in the use case of connected vehicle application, smart vehicles like NVIDIA self-driving cars have considerable computing capacity that can be shared during idle period, e.g. when temporarily parking besides the road. As such, edge service incorporates more choices of offloading servers that are dynamic and unpredictable. Solutions to efficiently utilise those resource will be attractive and beneficial.

Similarly, the rise of 5G networks and other networking developments will enable new possibilities that we may take advantage of. Faster data transmission and better networking coverage promised by those techniques are able to expand the feasible scenarios that edge computing can fit. Then it brings up the challenge of server placement and service platform design that can apply generally and expand automatically without requiring too much manual effort.

Another important and challenging aspect of the field is that, since edge computing distributes the service, task scheduling among multiple servers require dedicated algorithm and policy design to account for the transmission latency and packet loss between servers. Comparing with cloud data center in which servers communicate via wired cables that are much faster and more reliable, mobile computing and communication systems mostly carry out server communications in wireless network. Outdoor scenarios make it even more challenging due to the crowded users and interferences. Edge solution must tackle those challenges by including robust mechanism design in the task scheduling and other parts that matter.

# References


[1] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill, "Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity," in *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, 2004, pp. 39–47.

[2] I. Hadžić, Y. Abe, and H. C. Woithe, "Edge computing in the epc: a reality check," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, 2017, pp. 1–10.

[3] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *2008 10th ieee international conference on high performance computing and communications*. Ieee, 2008, pp. 825–830.

[4] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, vol. 1, 2015.

[5] C. Westphal, "Challenges in networking to support augmented reality and virtual reality," *IEEE ICNC*, 2017.

[6] "Company History — Akamai." [Online]. Available: https://www.akamai.com/uk/en/about/company-history.jsp

[7] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain, "Overlay networks: An akamai perspective," *Advanced Content Delivery, Streaming, and Cloud Services*, vol. 51, no. 4, pp. 305–328, 2014.

[8] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," 2015.

[9] "Wi-Fi 6 Explained: The Next Generation of Wi-Fi - TechSpot." [Online]. Available: https://www.techspot.com/article/1769-wi-fi-6-explained/

[10] Z. Xu, X. Li, X. Zhao, M. H. Zhang, and Z. Wang, "Dsrc versus 4g-lte for connected vehicle applications: A study on field experiments of vehicular communication performance," *Journal of Advanced Transportation*, vol. 2017, 2017.

[11] Wikipedia contributors, "Ieee 802.11ax — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=IEEE_802.11ax&oldid=954932604, 2020, [Online; accessed 8-May-2020].

[12] "Intel Unveils Its Automated Driving Compute Challenger To Nvidia." [Online]. Available: https://www.forbes.com/sites/samabuelsamid/2018/01/08/intel-unveils-its-automated-driving-compute-challenger-to-nvidia/{#}7c37d99e40ad

[13] Wikipedia contributors, "Pokémon go live events — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Pok%C3%A9mon_Go_live_events&oldid=955208174, 2020, [Online; accessed 8-May-2020].

[14] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13–16.

[16] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2014, pp. 325–329.

[17] F. Computing *et al.*, "Fog computing and the internet of things: Extend the cloud to where the things are," in *Technical Report*. Cisco Systems, 2016.

[18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[19] G. Adomavicius, J. C. Bockstedt, S. P. Curley, and J. Zhang, "Do recommender systems manipulate consumer preferences? a study of anchoring effects," *Information Systems Research*, vol. 24, no. 4, pp. 956–975, 2013.

[20] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4, pp. 333–344, 2006.

[21] Wikipedia contributors, "Anchoring (cognitive bias) — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Anchoring_(cognitive_bias)&oldid=949120215, 2020, [Online; accessed 8-May-2020].

[22] V. Cisco, "White paper: Cisco vni forecast and methodology, 2015-2020," *Retrieved November*, vol. 25, p. 2016, 2016.

[23] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, pp. 1–14.

[24] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[25] F. Qiu and J. Cho, "Automatic identification of user interest for personalized search," in *Proceedings of the 15th international conference on World Wide Web*, 2006, pp. 727–736.

[26] F. Orlandi, J. Breslin, and A. Passant, "Aggregated, interoperable and multi-domain user profiles for the social web," in *Proceedings of the 8th International Conference on Semantic Systems*, 2012, pp. 41–48.

[27] Z. Yu, X. Zhou, Y. Hao, and J. Gu, "Tv program recommendation for multiple viewers based on user profile merging," *User modeling and user-adapted interaction*, vol. 16, no. 1, pp. 63–82, 2006.

[28] K.-W. Hwang, D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, V. Misra, K. K. Ramakrishnan, and D. F. Swayne, "Leveraging video viewing patterns for optimal content placement," in *International Conference on Research in Networking*. Springer, 2012, pp. 44–58.

[29] M. S. ElBamby, M. Bennis, W. Saad, and M. Latva-Aho, "Content-aware user clustering and caching in wireless small cell networks," in *2014 11th International Symposium on Wireless Communications Systems (ISWCS)*. IEEE, 2014, pp. 945–949.

[30] Z. Chen and M. Kountouris, "Cache-enabled small cell networks with local user interest correlation," in *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2015, pp. 680–684.

[31] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proceedings of the second edition of the ICN workshop on Information-centric networking*, 2012, pp. 55–60.

[32] S. Guo, H. Xie, and G. Shi, "Collaborative forwarding and caching in content centric networks," in *International Conference on Research in Networking*. Springer, 2012, pp. 41–55.

[33] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, pp. 1–6, 2011.

[34] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[35] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[36] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information centric networking (icn)," in *SimuTools*, vol. 7. ICST, 2014, pp. 66–75.

[37] Wikipedia contributors, "Zipf's law — Wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Zipf%27s_law&oldid=955406626, 2020, [Online; accessed 8-May-2020].

[38] C. F. C. Solutions, "Unleash the power of the internet of things," *Cisco Systems Inc*, 2015.

[39] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in *2016 Cloudification of the Internet of Things (CIoT)*. IEEE, 2016, pp. 1–6.

[40] L. Ramaswamy, L. Liu, and A. Iyengar, "Cache clouds: Cooperative caching of dynamic documents in edge networks," in *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. IEEE, 2005, pp. 229–238.

[41] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Computer Networks*, vol. 57, no. 16, pp. 3128–3141, 2013.

[42] "Broadband by the Numbers — NCTA — The Internet & Television Association." [Online]. Available: https://www.ncta.com/broadband-by-the-numbers

[43] "Production Assistants (APAS) — Bosch Rexroth AG." [Online]. Available: https://www.boschrexroth.com/en/xc/products/product-groups/production-assistants-apas/template-overview-9

[44] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, 2012, pp. 29–36.

[45] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, 2013, pp. 15–20.

[46] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing*, 2012, pp. 145–154.

[47] Y. Li and W. Wang, "Can mobile cloudlets support mobile applications?" in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1060–1068.

[48] G. Pierre and M. Van Steen, "Globule: a collaborative content delivery network," *IEEE Communications Magazine*, vol. 44, no. 8, pp. 127–133, 2006.

[49] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing," *IEEE Signal Processing Magazine*, vol. 28, no. 3, pp. 59–69, 2011.

[50] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[51] L. Ramaswamy, L. Liu, and J. Zhang, "Efficient formation of edge cache groups for dynamic content delivery," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. IEEE, 2006, pp. 43–43.

[52] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," *ACM SIGARCH Computer Architecture News*, vol. 16, no. 2, pp. 280–298, 1988.

[53] J. Rajahalme, M. Särelä, K. Visala, and J. Riihijärvi, "On name-based inter-domain routing," *Computer Networks*, vol. 55, no. 4, pp. 975–986, 2011.

[54] G. Rossini and D. Rossi, "Coupling caching and forwarding: Benefits, analysis, and implementation," in *Proceedings of the 1st ACM Conference on Information-Centric Networking*, 2014, pp. 127–136.

[55] A. J. Hawkins, "California green lights fully driverless cars for testing on public roads," *The Verge*, vol. 26, 2018.

[56] "Shanghai allows autonomous tests - Business - Chinadaily.com.cn." [Online]. Available: http://www.chinadaily.com.cn/business/motoring/2017-11/13/content{_}34469664.htm

[57] M. Faezipour, M. Nourani, A. Saeed, and S. Addepalli, "Progress and challenges in intelligent vehicle area networks," *Communications of the ACM*, vol. 55, no. 2, pp. 90–100, 2012.

[58] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE world forum on internet of things (WF-IoT)*. IEEE, 2014, pp. 241–246.

[59] A. Nordrum, "Autonomous driving experts weigh 5g cellular network against dedicated short range communications," *IEEE Spectrum, Cars That Think*, 2016.

[60] "Connected car battle lines are drawn – TU Automotive." [Online]. Available: https://www.tu-auto.com/connected-car-battle-lines-are-drawn/

[61] A. Filippi, K. Moerman, G. Daalderop, P. D. Alexander, F. Schober, and W. Pfliegl, "Ready to roll: Why 802.11 p beats lte and 5g for v2x," *WhitePaper by NXP Semiconductors, Cohda Wireless and Siemens*, 2016.

[62] X. Huang, R. Yu, J. Kang, Y. He, and Y. Zhang, "Exploring mobile edge computing for 5g-enabled software defined vehicular networks," *IEEE Wireless Communications*, vol. 24, no. 6, pp. 55–63, 2017.

[63] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, "On uncoordinated service placement in edge-clouds," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 41–48.

[64] D. Grewe, M. Wagner, M. Arumaithurai, I. Psaras, and D. Kutscher, "Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions," in *Proceedings of the Workshop on Mobile Edge Communications*, 2017, pp. 7–12.

[65] H. Qiu, F. Ahmad, R. Govindan, M. Gruteser, F. Bai, and G. Kar, "Augmented vehicular reality: Enabling extended vision for future vehicles," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*, 2017, pp. 67–72.

[66] H. Kim, X. Wu, J. L. Gabbard, and N. F. Polys, "Exploring head-up augmented reality interfaces for crash warning systems," in *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, 2013, pp. 224–227.

[67] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[68] "Inside Cruise's Bumpy Ride: The Limits of Self-Driving Cars — The Information." [Online]. Available: https://www.theinformation.com/articles/inside-cruises-bumpy-ride-the-limits-of-self-driving-cars

[69] "WayRay." [Online]. Available: https://wayray.com/navion

[70] H. S. Park, M. W. Park, K. H. Won, K.-H. Kim, and S. K. Jung, "In-vehicle ar-hud system to provide driving-safety information," *ETRI journal*, vol. 35, no. 6, pp. 1038–1047, 2013.

[71] H. Park and K.-h. Kim, "Efficient information representation method for driver-centered ar-hud system," in *International Conference of*

*Design, User Experience, and Usability.* Springer, 2013, pp. 393–400.

[72] M.-E. Computing, "European telecomm. standards inst.(etsi), 2014."

[73] S. C. Forum, "Small cell forum unveils operator research showing accelerating densification and enterprise deployments on road to 5g, 2017."

[74] "No technical challenge too big for small cells — Nokia Blog." [Online]. Available: https://www.nokia.com/blog/no-technical-challenge-big-small-cells/

[75] S. Engel, C. Kratzsch, and K. David, "Car2pedestrian-communication: Protection of vulnerable road users using smartphones," in *Advanced Microsystems for Automotive Applications 2013.* Springer, 2013, pp. 31–41.

[76] P. Zhou, W. Zhang, T. Braud, P. Hui, and J. Kangasharju, "Arve: Augmented reality applications in vehicle to edge networks," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 25–30.

[77] "Home - EdgeX Foundry." [Online]. Available: https://www.edgexfoundry.org/

[78] "PowerEdge C4140 Server : Rack Servers — Dell USA." [Online]. Available: https://www.dell.com/en-us/work/shop/povw/poweredge-c4140

[79] Q. Xiao, K. Xu, D. Wang, L. Li, and Y. Zhong, "Tcp performance over mobile networks in high-speed mobility scenarios," in *2014 IEEE 22nd International Conference on Network Protocols.* IEEE, 2014, pp. 281–286.

[80] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS).* IEEE, 2017, pp. 1796–1807.

[81] M. Abrash, "Latency–the sine qua non of ar and vr," *Blog post, Dec*, 2012.

[82] J. Licklider, "Memorandum for: Members and affiliates of the intergalactic computer network; topics for discussion at the forthcoming

meeting," *Washington, DC: Advanced Research Projects Agency. Retrieved April*, vol. 21, 2011.

[83] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[84] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Communications surveys & tutorials*, vol. 15, no. 3, pp. 1294–1313, 2012.

[85] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[86] S. Yu, R. Langar, and X. Wang, "A d2d-multicast based computation offloading framework for interactive applications," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[87] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[88] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[89] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 proceedings IEEE Infocom*. IEEE, 2012, pp. 945–953.

[90] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in *International Conference on Mobile Computing, Applications, and Services*. Springer, 2010, pp. 59–79.

[91] D. Wagner and D. Schmalstieg, "First steps towards handheld augmented reality," in *Seventh IEEE International Symposium on Wearable Computers, 2003. Proceedings.* IEEE, 2003, pp. 127–135.

[92] B. Shi, J. Yang, Z. Huang, and P. Hui, "Offloading guidelines for augmented reality applications on wearable devices," in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 1271–1274.

[93] P. Jain, J. Manweiler, and R. Roy Choudhury, "Overlay: Practical mobile augmented reality," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, 2015, pp. 331–344.

[94] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1801–1819, 2014.

[95] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proceedings of the 19th annual international conference on Mobile computing & networking*, 2013, pp. 203–205.

[96] B. Han, P. Hui, V. A. Kumar, M. V. Marathe, J. Shao, and A. Srinivasan, "Mobile data offloading through opportunistic communications and social participation," *IEEE Transactions on mobile computing*, vol. 11, no. 5, pp. 821–834, 2011.

[97] D. Chatzopoulos, K. Sucipto, S. Kosta, and P. Hui, "Video compression in the neighborhood: An opportunistic approach," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[98] K. Sucipto, D. Chatzopoulos, S. Kosta, and P. Hui, "Keep your nice friends close, but your rich friends closer—computation offloading using nfc," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[99] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[100] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks," *IEEE Signal Processing Magazine*, vol. 31, no. 6, pp. 45–55, 2014.

[101] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[102] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White paper, mobile-edge computing (MEC) industry initiative*, pp. 1089–7801, 2014.

[103] "Real-world impact of mobile edge computing (mec)," Tech. Rep., 2016. [Online]. Available: https://builders.intel.com/docs/networkbuilders/Real-world-impact-of-mobile-edge-computing-MEC.pdf

[104] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[105] Y. Li and W. Gao, "Code offload with least context migration in the mobile cloud," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1876–1884.

[106] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb, "Mobiscud: A fast moving personal cloud in the mobile network," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, 2015, pp. 19–24.

[107] W. Zhang, B. Han, and P. Hui, "On the networking challenges of mobile augmented reality," in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, 2017, pp. 24–29.

[108] "Vuforia: Market-Leading Enterprise AR — PTC." [Online]. Available: https://www.ptc.com/en/products/augmented-reality/vuforia

[109] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[110] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[111] G. Cookson and B. Pishue, "Inrix global traffic scorecard. inrix research," 2017.

[112] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1624–1639, 2011.

[113] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: a deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2014.

[114] R. L. Bertini, S. Hansen, A. Byrd, and T. Yin, "Experience implementing a user service for archived intelligent transportation systems data," *Transportation research record*, vol. 1917, no. 1, pp. 90–99, 2005.

[115] Y. Ding, C. Chen, S. Zhang, B. Guo, Z. Yu, and Y. Wang, "Greenplanner: Planning personalized fuel-efficient driving routes using multi-sourced urban data," in *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2017, pp. 207–216.

[116] J. Gao, Y. Shen, J. Liu, M. Ito, and N. Shiratori, "Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network," *arXiv preprint arXiv:1705.02755*, 2017.

[117] W. Genders and S. Razavi, "Using a deep reinforcement learning agent for traffic signal control," *arXiv preprint arXiv:1611.01142*, 2016.

[118] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, 2016.

[119] X. Liang, X. Du, G. Wang, and Z. Han, "Deep reinforcement learning for traffic light control in vehicular networks," *arXiv preprint arXiv:1803.11115*, 2018.

[120] A. Barisone, D. Giglio, R. Minciardi, and R. Poggi, "A macroscopic traffic model for real-time optimization of signalized urban areas," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 1. IEEE, 2002, pp. 900–903.

[121] D. Krajzewicz, E. Brockfeld, J. Mikat, J. Ringel, C. Rössel, W. Tuchscheerer, P. Wagner, and R. Wösler, "Simulation of modern traffic lights control systems using the open source traffic simulation

sumo," in *Proceedings of the 3rd Industrial Simulation Conference 2005*. EUROSIS-ETI, 2005, pp. 299–302.

[122] J. Mikat, E. Brockfeld, and P. Wagner, "Agent based traffic signals regulating flow on a basic grid."

[123] B. Abdulhai, R. Pringle, and G. J. Karakoulas, "Reinforcement learning for true adaptive traffic signal control," *Journal of Transportation Engineering*, vol. 129, no. 3, pp. 278–285, 2003.

[124] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[125] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[126] "keras-team/keras: Deep Learning for humans." [Online]. Available: https://github.com/keras-team/keras

[127] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, 2012.

[128] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

TIETOJENKÄSITTELYTIETEEN OSASTO
PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki, FINLAND

JULKAISUSARJA **A**

SERIES OF PUBLICATIONS **A**

Reports are available on the e-thesis site of the University of Helsinki.

A-2015-1 L. Wang: Content, Topology and Cooperation in In-network Caching. 190 pp. (Ph.D. Thesis)

A-2015-2 T. Niinimäki: Approximation Strategies for Structure Learning in Bayesian Networks. 64+93 pp. (Ph.D. Thesis)

A-2015-3 D. Kempa: Efficient Construction of Fundamental Data Structures in Large-Scale Text Indexing. 68+88 pp. (Ph.D. Thesis)

A-2015-4 K. Zhao: Understanding Urban Human Mobility for Network Applications. 62+46 pp. (Ph.D. Thesis)

A-2015-5 A. Laaksonen: Algorithms for Melody Search and Transcription. 36+54 pp. (Ph.D. Thesis)

A-2015-6 Y. Ding: Collaborative Traffic Offloading for Mobile Systems. 223 pp. (Ph.D. Thesis)

A-2015-7 F. Fagerholm: Software Developer Experience: Case Studies in Lean-Agile and Open Source Environments. 118+68 pp. (Ph.D. Thesis)

A-2016-1 T. Ahonen: Cover Song Identification using Compression-based Distance Measures. 122+25 pp. (Ph.D. Thesis)

A-2016-2 O. Gross: World Associations as a Language Model for Generative and Creative Tasks. 60+10+54 pp. (Ph.D. Thesis)

A-2016-3 J. Määttä: Model Selection Methods for Linear Regression and Phylogenetic Reconstruction. 44+73 pp. (Ph.D. Thesis)

A-2016-4 J. Toivanen: Methods and Models in Linguistic and Musical Computational Creativity. 56+8+79 pp. (Ph.D. Thesis)

A-2016-5 K. Athukorala: Information Search as Adaptive Interaction. 122 pp. (Ph.D. Thesis)

A-2016-6 J.-K. Kangas: Combinatorial Algorithms with Applications in Learning Graphical Models. 66+90 pp. (Ph.D. Thesis)

A-2017-1 Y. Zou: On Model Selection for Bayesian Networks and Sparse Logistic Regression. 58+61 pp. (Ph.D. Thesis)

A-2017-2 Y.-T. Hsieh: Exploring Hand-Based Haptic Interfaces for Mobile Interaction Design. 79+120 pp. (Ph.D. Thesis)

A-2017-3 D. Valenzuela: Algorithms and Data Structures for Sequence Analysis in the Pan-Genomic Era. 74+78 pp. (Ph.D. Thesis)

A-2017-4 A. Hellas: Retention in Introductory Programming. 68+88 pp. (Ph.D. Thesis)

A-2017-5 M. Du: Natural Language Processing System for Business Intelligence. 78+72 pp. (Ph.D. Thesis)

A-2017-6 A. Kuosmanen: Third-Generation RNA-Sequencing Analysis: Graph Alignment and Transcript Assembly with Long Reads. 64+69 pp. (Ph.D. Thesis)

A-2018-1 M. Nelimarkka: Performative Hybrid Interaction: Understanding Planned Events across Collocated and Mediated Interaction Spheres. 64+82 pp. (Ph.D. Thesis)

A-2018-2 E. Peltonen: Crowdsensed Mobile Data Analytics. 100+91 pp. (Ph.D. Thesis)

A-2018-3 O. Barral: Implicit Interaction with Textual Information using Physiological Signals. 72+145 pp. (Ph.D. Thesis)

A-2018-4 I. Kosunen: Exploring the Dynamics of the Biocybernetic Loop in Physiological Computing. 91+161 pp. (Ph.D. Thesis)

A-2018-5 J. Berg: Solving Optimization Problems via Maximum Satisfiability: Encodings and Re-Encodings. 86+102 pp. (Ph.D. Thesis)

A-2018-6 J. Pyykkö: Online Personalization in Exploratory Search. 101+63 pp. (Ph.D. Thesis)

A-2018-7 L. Pivovarova: Classification and Clustering in Media Monitoring: from Knowledge Engineering to Deep Learning. 78+56 pp. (Ph.D. Thesis)

A-2019-1 K. Salo: Modular Audio Platform for Youth Engagement in a Museum Context. 97+78 pp. (Ph.D. Thesis)

A-2019-2 A. Koski: On the Provisioning of Mission Critical Information Systems based on Public Tenders. 96+79 pp. (Ph.D. Thesis)

A-2019-3 A. Kantosalo: Human-Computer Co-Creativity - Designing, Evaluating and Modelling Computational Collaborators for Poetry Writing. 74+86 pp. (Ph.D. Thesis)

A-2019-4 O. Karkulahti: Understanding Social Media through Large Volume Measurements. 116 pp. (Ph.D. Thesis)

A-2019-5 S. Yaman: Initiating the Transition towards Continuous Experimentation: Empirical Studies with Software Development Teams and Practitioners. 81+90 pp. (Ph.D. Thesis)

A-2019-6 N. Mohan: Edge Computing Platforms and Protocols. 87+69 pp. (Ph.D. Thesis)

A-2019-7 I. Järvinen: Congestion Control and Active Queue Management During Flow Startup. 87+48 pp. (Ph.D. Thesis)

A-2019-8 J. Leinonen: Keystroke Data in Programming Courses. 56+53 pp. (Ph.D. Thesis)

A-2019-9 T. Talvitie: Counting and Sampling Directed Acyclic Graphs for Learning Bayesian Networks. 70+54 pp. (Ph.D. Thesis)

A-2019-10 J. Toivonen: Modeling and Learning Monomeric and Dimeric Transcription Factor Binding Motifs. 61+109 pp. (Ph.D. Thesis)

A-2019-11 S. Hemminki: Advances in Motion Sensing on Mobile Devices. 113+89 pp. (Ph.D. Thesis)

A-2019-12 P. Saikko: Implicit Hitting Set Algorithms for Constraint Optimization. 70+54 pp. (Ph.D. Thesis)

A-2020-1 J. Leppä-aho: Methods for Learning Directed and Undirected Graphical Models. 50+84 pp. (Ph.D. Thesis)